**Danh sách đặc**

**Các phép toán trên danh sách đặc các số nguyên**

**Khai báo**

#define MaxLength 100

typedef struct {

int Elements[MaxLength];

int Last;

} List;

Tên hàm: **makenullList()**

void makenullList(List\* pL) {

pL->Last = 0;

}

Tên hàm (function name): **member()**

int member(int x, List L) {

for (int i = 0; i < L.Last; i++) {

if (L.Elements[i] == x) {

return 1;

}

} return 0;

}

Tên hàm (function name): **sort()**

void sort(List \*pL) {

int i, j;

for (i = 0; i < pL->Last - 1; i++) {

for (j = i + 1; j < pL->Last; j++) {

if (pL->Elements[i] > pL->Elements[j]) {

int temp = pL->Elements[i];

pL->Elements[i] = pL->Elements[j];

pL->Elements[j] = temp;

}

}

}

}

Tên hàm (function name): **unionSet()**

void unionSet(List L1, List L2, List \*pL) {

makenullList(pL);

for (int i = 0; i < L1.Last; i++) {

insertSet(L1.Elements[i], pL);

}

for (int j = 0; j < L2.Last; j++) {

if (!member(L2.Elements[j], \*pL)) {

insertSet(L2.Elements[j], pL);

}

}

}

Tên hàm (function name**): deleteList()**

void deleteList(Position p, List \*pL) {

if (p < 1 || p > pL->Last) {

printf("Vi tri khong hop le\n");

return;

}

if (pL->Last == 0) {

printf("Danh sach rong!\n");

return;

}

for (int Q = p - 1; Q < pL->Last - 1; Q++) {

pL->Elements[Q] = pL->Elements[Q + 1];

}

pL->Last--;

}

Tên hàm (function name): **removeAll()**

void removeAll(int x, List \*pL) {

int pos;

while ((pos = locate(x, \*pL)) != pL->Last + 1) { // Nếu tìm thấy x

deleteList(pos, pL); // Xóa phần tử tại vị trí tìm thấy

}

}

Tên hàm (function name): **intersection()**

void intersection(List L1, List L2, List \*pL) {

makenullList(pL);

for (int i = 0; i < L1.Last; i++) {

ElementType x = L1.Elements[i];

if (member(x, L2)) {

insertSet(x, pL);

}

}

}

Tên hàm (function name): **copyEvenNumbers(**)

void copyEvenNumbers(List L1, List \*pL2) {

makenullList(pL2);

for (int i = 0; i < L1.Last; i++) {

if (L1.Elements[i] % 2 == 0) {

insertList(L1.Elements[i], pL2->Last + 1, pL2);

}

}

}

Tên hàm (function name): **insertSet()**

void insertSet(int x, List \*pL) {

if (!member(x, \*pL)) {

pL->Elements[pL->Last] = x;

pL->Last++;

}

}

Tên hàm (function name): **insertList()**

void insertList(int x, int p, List \*pL) {

if (p < 1 || p > pL->Last + 1) {

printf("Vị trí không hợp lệ\n");

return;

}

for (int i = pL->Last; i >= p; i--) {

pL->Elements[i] = pL->Elements[i - 1];

}

pL->Elements[p - 1] = x;

pL->Last++;

}

Tên hàm (function name): **locate()**

int locate(int x, List L) {

for (int i = 0; i < L.Last; i++) {

if (L.Elements[i] == x) {

return i + 1;

}

}

return L.Last + 1;

}

Tên hàm (function name): **normalize()**

void normalize(List \*pL) {

for (int i = 0; i < pL->Last; i++) {

for (int j = i + 1; j < pL->Last; ) {

if (pL->Elements[i] == pL->Elements[j]) {

for (int k = j; k < pL->Last - 1; k++) {

pL->Elements[k] = pL->Elements[k + 1];

}

pL->Last--;

} else {

j++;

}

}

}

}

Tên hàm (function name): **printOddNumbers()**

void printOddNumbers(List L) {

for (int i = 0; i < L.Last; i++) {

if (L.Elements[i] % 2 != 0) {

printf("%d ", L.Elements[i]);

}

}

printf("\n");

}

Tên hàm (function name): **printList()**

void printList(List L) {

for (int i = 0; i < L.Last; i++) {

printf("%d ", L.Elements[i]);

}

printf("\n");

}

Tên hàm (function name): **getAvg()**

float getAvg(List L) {

if (L.Last == 0) {

return -10000.0000;

}

float sum = 0;

for (int i = 0; i < L.Last; i++) {

sum += L.Elements[i];

}

return sum / L.Last;

}

Tên hàm (function name): **readList()**

void readList(List \*pL) {

int n, x;

makenullList(pL);

scanf("%d", &n);

for (int i = 0; i < n; i++) {

scanf("%d", &x);

insertList(x, pL->Last + 1, pL);

}

}

Tên hàm (function name): **readSet()**

void readSet(List \*pL) {

int n, x;

makenullList(pL);

scanf("%d", &n);

for (int i = 0; i < n; i++) {

scanf("%d", &x);

if (!member(x, \*pL)) {

insertSet(x, pL);

}

}

}

Tên hàm (function name): **difference()**

void difference(List L1, List L2, List \*pL) {

makenullList(pL);

for (int i = 0; i < L1.Last; i++) {

if (!member(L1.Elements[i], L2)) { trong L2

insertSet(L1.Elements[i], pL);

}

}

}

Tên hàm (function name): erase()

void erase(int x, List \*pL) {

int p = locate(x, \*pL);

if (p != -1) { // Kiểm tra nếu p là một vị trí hợp lệ

deleteList(p, pL);

}

}

**Áp dụng danh sách đặc các số nguyên**

**Bằng cách sử dụng những phép toán chuẩn trên danh sách, hãy viết chương trình cho phép liệt kê các số lẻ, số chẵn trong danh sách**

#include <stdio.h>

#include <stdlib.h>

#define MaxLength 100

typedef struct {

int Elements[MaxLength];

int Last;

} List;

// Các hàm đã được định nghĩa trước đó

void makenullList(List \*pL) {

pL->Last = 0;

}

int member(int x, List L) {

for (int i = 0; i < L.Last; i++) {

if (L.Elements[i] == x) {

return 1;

}

}

return 0;

}

void insertSet(int x, List \*pL) {

if (pL->Last < MaxLength) {

pL->Elements[pL->Last] = x;

pL->Last++;

}

}

void printOddNumbers(List L) {

for (int i = 0; i < L.Last; i++) {

if (L.Elements[i] % 2 != 0) {

printf("%d ", L.Elements[i]);

}

}

printf("\n");

}

void printEvenNumbers(List L) {

for (int i = 0; i < L.Last; i++) {

if (L.Elements[i] % 2 == 0) {

printf("%d ", L.Elements[i]);

}

}

printf("\n");

}

void printList(List L) {

for (int i = 0; i < L.Last; i++) {

printf("%d ", L.Elements[i]);

}

printf("\n");

}

void readList(List \*pL) {

int n, x;

makenullList(pL);

scanf("%d", &n);

for (int i = 0; i < n; i++) {

scanf("%d", &x);

insertSet(x, pL);

}

}

int main() {

List L;

readList(&L);

printList(L);

printOddNumbers(L);

printEvenNumbers(L);

return 0;

}

**Bằng cách sử dụng những phép toán chuẩn trên danh sách, hãy viết chương trình cho phép tính trung bình cộng các phần tử trong danh sách**

#include <stdio.h>

#include <stdlib.h>

#define MaxLength 100

typedef struct {

int Elements[MaxLength];

int Last;

} List;

void makenullList(List \*pL) {

pL->Last = 0;

}

void insertSet(int x, List \*pL) {

if (pL->Last < MaxLength) {

pL->Elements[pL->Last] = x;

pL->Last++;

}

}

void readList(List \*pL) {

int n, x;

makenullList(pL);

scanf("%d", &n);

for (int i = 0; i < n; i++) {

scanf("%d", &x);

insertSet(x, pL);

}

}

void printList(List L) {

for (int i = 0; i < L.Last; i++) {

printf("%d ", L.Elements[i]);

}

printf("\n");

}

double average(List L) {

if (L.Last == 0) return 0;

int sum = 0;

for (int i = 0; i < L.Last; i++) {

sum += L.Elements[i];

}

return (double)sum / L.Last;

}

int main() {

List L;

readList(&L);

printList(L);

double avg = average(L);

printf("%.3f\n", avg);

return 0;

}

**Bằng cách sử dụng những phép toán chuẩn trên danh sách, hãy viết chương trình cho phép tìm tập hợp hợp của 2 danh sách biểu diễn tập hợp**

#include <stdio.h>

#include <stdlib.h>

#define MaxLength 100

typedef struct {

int Elements[MaxLength];

int Last;

} List;

// Hàm khởi tạo danh sách

void makenullList(List \*pL) {

pL->Last = 0;

}

// Hàm thêm phần tử vào danh sách nếu chưa có

void insertUnique(int x, List \*pL) {

for (int i = 0; i < pL->Last; i++) {

if (pL->Elements[i] == x) {

return; // Nếu phần tử đã có, không thêm

}

}

if (pL->Last < MaxLength) {

pL->Elements[pL->Last] = x;

pL->Last++;

}

}

// Hàm đọc danh sách từ bàn phím

void readList(List \*pL) {

int n, x;

makenullList(pL);

scanf("%d", &n);

for (int i = 0; i < n; i++) {

scanf("%d", &x);

insertUnique(x, pL);

}

}

// Hàm hiển thị danh sách

void printList(List L) {

for (int i = 0; i < L.Last; i++) {

printf("%d ", L.Elements[i]);

}

printf("\n");

}

// Hàm tìm tập hợp hợp của 2 tập hợp

void unionSets(List A, List B, List \*C) {

makenullList(C);

// Thêm phần tử từ A

for (int i = 0; i < A.Last; i++) {

insertUnique(A.Elements[i], C);

}

// Thêm phần tử từ B

for (int i = 0; i < B.Last; i++) {

insertUnique(B.Elements[i], C);

}

}

int main() {

List A, B, C;

// Nhập tập hợp A

readList(&A);

printList(A);

// Nhập tập hợp B

readList(&B);

printList(B);

// Tìm tập hợp hợp

unionSets(A, B, &C);

// printf("Tập hợp hợp của A và B: ");

printList(C);

return 0;

}

Bằng cách sử dụng những phép toán chuẩn trên danh sách, hãy viết chương trình cho phéptìm hiệu của 2 danh sách biểu diễn tập hợp

#include <stdio.h>

#define Maxlength 300

typedef struct {

int Elements[Maxlength];

int Last;

} List;

// Khai báo trước hàm member

int member(int x, List L);

void makenullList(List \*pL) {

pL->Last = 0;

}

void insertSet(int x, List \*pL) {

// Kiểm tra xem phần tử đã tồn tại trong danh sách chưa

if (!member(x, \*pL)) {

pL->Elements[pL->Last] = x;

pL->Last++;

}

}

void readSet(List \*pL) {

makenullList(pL);

int n, x, i;

scanf("%d", &n);

for (i = 0; i < n; i++) {

scanf("%d", &x);

insertSet(x, pL);

}

}

void printList(List L) {

int i;

for (i = 0; i < L.Last; i++)

printf("%d ", L.Elements[i]);

printf("\n");

}

int member(int x, List L) {

int i;

for (i = 0; i < L.Last; i++) {

if (L.Elements[i] == x) return 1; // Nếu tìm thấy, trả về 1

}

return 0; // Nếu không tìm thấy, trả về 0

}

void difference(List L1, List L2, List \*pL) {

makenullList(pL);

int i;

for (i = 0; i < L1.Last; i++) {

if (!member(L1.Elements[i], L2)) {

insertSet(L1.Elements[i], pL); // Thêm vào pL nếu không có trong L2

}

}

}

int main() {

List L1, L2, L;

readSet(&L1);

readSet(&L2);

printList(L1);

printList(L2);

difference(L1, L2, &L);

printList(L);

return 0;

}

**Bằng cách sử dụng những phép toán chuẩn trên danh sách đã cho trong thư viện AListLib.c, hãy viết chương trình cho phép sắp xếp danh sách tăng dần**

#include <stdio.h>

#include "AListLib.c"

#define MaxLength 100

// Sử dụng kiểu dữ liệu khác để tránh xung đột với AListLib

typedef struct {

int Elements[MaxLength];

int Last;

} MyList;

void makenullMyList(MyList \*pL) {

pL->Last = 0;

}

void insertToMyList(int x, MyList \*pL) {

if (pL->Last < MaxLength) {

pL->Elements[pL->Last] = x;

pL->Last++;

}

}

void readMyList(MyList \*pL) {

int n, x;

makenullMyList(pL);

scanf("%d", &n);

for (int i = 0; i < n; i++) {

scanf("%d", &x);

insertToMyList(x, pL);

}

}

void printMyList(MyList L) {

for (int i = 0; i < L.Last; i++) {

printf("%d ", L.Elements[i]);

}

printf("\n");

}

void sortMyList(MyList \*pL) {

for (int i = 0; i < pL->Last - 1; i++) {

for (int j = 0; j < pL->Last - i - 1; j++) {

if (pL->Elements[j] > pL->Elements[j + 1]) {

// Hoán đổi

int temp = pL->Elements[j];

pL->Elements[j] = pL->Elements[j + 1];

pL->Elements[j + 1] = temp;

}

}

}

}

int main() {

MyList L;

readMyList(&L);

printMyList(L);

sortMyList(&L);

printMyList(L);

return 0;

}

**Bằng cách sử dụng những phép toán chuẩn trên danh sách, hãy viết chương trình cho phép tìm tập hợp giao của 2 danh sách biểu diễn tập hợp**

#include <stdio.h>

#include "AListLib.c" // Đảm bảo thư viện này chứa định nghĩa cho các hàm cần thiết

int member(int x, List L) {

for (int i = 0; i < L.Last; i++) {

if (L.Elements[i] == x) {

return 1; // Có thuộc

}

}

return 0; // Không thuộc

}

void insertSet(int x, List \*pL) {

if (!member(x, \*pL)) {

pL->Elements[pL->Last] = x; // Thêm phần tử vào cuối danh sách

pL->Last++; // Tăng kích thước danh sách

}

}

void readSet(List \*pL) {

int n, x;

makenullList(pL); // Khởi tạo danh sách rỗng

scanf("%d", &n); // Nhập số phần tử của tập hợp

for (int i = 0; i < n; i++) {

scanf("%d", &x); // Nhập phần tử x

insertSet(x, pL); // Chèn x vào tập hợp

}

}

void printList(List L) {

for (int i = 0; i < L.Last; i++) {

printf("%d ", L.Elements[i]); // Hiển thị phần tử tại chỉ số i

}

printf("\n"); // Xuống dòng sau khi in xong

}

void intersection(List L1, List L2, List \*pL) {

makenullList(pL); // Khởi tạo danh sách kết quả là rỗng

for (int i = 0; i < L1.Last; i++) {

if (member(L1.Elements[i], L2)) {

insertSet(L1.Elements[i], pL); // Chèn phần tử vào danh sách kết quả

}

}

}

int main() {

List L1, L2, result;

readSet(&L1);

printList(L1);

readSet(&L2);

printList(L2);

intersection(L1, L2, &result);

printList(result);

return 0;

}

**Hãy hoàn chỉnh chương trình trên bằng cách điền các lệnh cần thiết vào dấu ...  để tạo thành 1 chương trình cho phép nhập và hiển thị 1 danh sách từ bàn phím.**

readList(&L);

printList(L);

**Bằng cách sử dụng những phép toán chuẩn trên danh sách đã cho trong thư viện AListLib.c, hãy viết chương trình nhập một danh sách từ bàn phím và cho phép xóa phần tử x xuất hiện đầu tiên trong danh sách**

#include <stdio.h>

#define MaxLength 100

typedef struct {

int Elements[MaxLength]; // Mảng chứa các phần tử

int Last; // Số lượng phần tử hiện tại trong danh sách

} List;

// Hàm khởi tạo danh sách rỗng

void makenullList(List \*pL) {

pL->Last = 0;

}

// Hàm kiểm tra phần tử x có trong danh sách hay không

int member(int x, List L) {

for (int i = 0; i < L.Last; i++) {

if (L.Elements[i] == x) {

return 1;

}

}

return 0;

}

// Hàm thêm phần tử x vào cuối danh sách

void insertSet(int x, List \*pL) {

if (pL->Last < MaxLength) {

pL->Elements[pL->Last] = x;

pL->Last++;

}

}

// Hàm tìm vị trí xuất hiện đầu tiên của phần tử x trong danh sách

int locate(int x, List L) {

for (int i = 0; i < L.Last; i++) {

if (L.Elements[i] == x) {

return i + 1; // Trả về vị trí (thêm 1 để chuyển sang chỉ số bắt đầu từ 1)

}

}

return L.Last + 1; // Nếu không tìm thấy, trả về L.Last + 1

}

// Hàm xóa phần tử ở vị trí p trong danh sách

void deleteList(int p, List \*pL) {

if (p < 1 || p > pL->Last) {

return;

}

for (int Q = p - 1; Q < pL->Last - 1; Q++) {

pL->Elements[Q] = pL->Elements[Q + 1];

}

pL->Last--;

}

// Hàm xóa phần tử x xuất hiện đầu tiên trong danh sách

void erase(int x, List \*pL) {

int p = locate(x, \*pL); // Tìm vị trí xuất hiện đầu tiên của x trong danh sách

if (p <= pL->Last) { // Kiểm tra nếu p là một vị trí hợp lệ

deleteList(p, pL); // Xóa phần tử ở vị trí p

}

}

// Hàm nhập danh sách từ bàn phím

void readList(List \*pL) {

int n, x;

makenullList(pL); // Khởi tạo danh sách rỗng

scanf("%d", &n); // Nhập số phần tử của danh sách

for (int i = 0; i < n; i++) {

scanf("%d", &x); // Nhập phần tử x

insertSet(x, pL); // Chèn x vào danh sách

}

}

// Hàm hiển thị danh sách

void printList(List L) {

for (int i = 0; i < L.Last; i++) {

printf("%d ", L.Elements[i]); // Hiển thị phần tử tại chỉ số i

}

printf("\n"); // Xuống dòng sau khi in xong

}

// Hàm main

int main() {

List L;

readList(&L); // Nhập danh sách

printList(L); // Hiển thị danh sách

int x;

scanf("%d", &x); // Nhập giá trị x cần xóa

erase(x, &L); // Xóa phần tử x xuất hiện đầu tiên

printList(L); // Hiển thị danh sách sau khi xóa

return 0;

}

**Bằng cách sử dụng những phép toán chuẩn trên danh sách, hãy viết chương trình cho phép liệt kê các số chẵn và tính trung bình cộng các phần tử là số chẵn trong danh sách**

#include <stdio.h>

#define MaxLength 100

typedef struct {

int Elements[MaxLength]; // Mảng chứa các phần tử

int Last; // Số lượng phần tử hiện tại trong danh sách

} List;

void makenullList(List \*pL) {

pL->Last = 0;

}

void insertSet(int x, List \*pL) {

pL->Elements[pL->Last] = x;

pL->Last++;

}

void readList(List \*pL) {

int n, x;

makenullList(pL); // Khởi tạo danh sách rỗng

scanf("%d", &n); // Nhập số phần tử của danh sách

for (int i = 0; i < n; i++) {

scanf("%d", &x); // Nhập phần tử x

insertSet(x, pL); // Chèn x vào danh sách

}

}

void copyEvenNumbers(List L1, List \*pL2) {

makenullList(pL2); // Khởi tạo danh sách kết quả rỗng

for (int i = 0; i < L1.Last; i++) {

if (L1.Elements[i] % 2 == 0) {

insertSet(L1.Elements[i], pL2);

}

}

}

void printList(List L) {

for (int i = 0; i < L.Last; i++) {

printf("%d ", L.Elements[i]);

}

printf("\n");

}

float getAvg(List L) {

if (L.Last == 0) {

return -10000.0000; // Trả về -10000.0000 nếu danh sách rỗng

}

float sum = 0;

for (int i = 0; i < L.Last; i++) {

sum += L.Elements[i];

}

return sum / L.Last; // Tính trung bình cộng

}

int main() {

List L, EvenList;

readList(&L);

printList(L);

copyEvenNumbers(L, &EvenList);

printList(EvenList);

float avg = getAvg(EvenList);

printf("%.3f\n", avg);

return 0;

}

**Bằng cách sử dụng những phép toán chuẩn trên danh sách, hãy viết chương trình cho phép xóa tất cả các phần tử x trong danh sách**

#include <stdio.h>

#include "AListLib.c"

void readList(List \*pL) {

int n, x;

makenullList(pL);

scanf("%d", &n);

for (int i = 0; i < n; i++) {

scanf("%d", &x); // Nhập phần tử x

insertList(x, pL->Last + 1, pL);

}

}

void printList(List L) {

for (int i = 0; i < L.Last; i++) {

printf("%d ", L.Elements[i]);

}

printf("\n");

}

void eraseAll(int x, List \*pL) {

int pos;

while ((pos = locate(x, \*pL)) != pL->Last + 1) { // Tìm vị trí xuất hiện đầu tiên của x

deleteList(pos, pL); // Xóa phần tử ở vị trí tìm được

}

}

int main() {

List L;

int x;

readList(&L);

printList(L);

scanf("%d", &x);

eraseAll(x, &L);

printList(L);

return 0;

}

Các phép toán của danh sách sinh viên

**Giả sử khai báo kiểu DanhSach đã được định nghĩa.  Hãy viết 1 hàm hiển thị**[**danh sách sinh viên**](https://else.ctu.edu.vn/mod/quiz/view.php?id=9015)

**Tên hàm: hienthi()**

void hienthi(DanhSach L) {

for (int i = 0; i < L.n; i++) {

float tongDiem = L.A[i].DiemLT + L.A[i].DiemTH1 + L.A[i].DiemTH2;

printf("%s - %s - %.2f - %.2f - %.2f - %.2f\n",

L.A[i].MSSV,

L.A[i].HoTen,

L.A[i].DiemLT,

L.A[i].DiemTH1,

L.A[i].DiemTH2,

tongDiem);

}

}

**Giả sử khai báo kiểu DanhSach đã được định nghĩa.  Hãy viết 1 hàm tìm vị trí sinh viên có mã số x cho trước**

**Tên hàm: tim()**

int tim(const char \*x, DanhSach L) {

for (int i = 0; i < L.n; i++) {

// So sánh mã số sinh viên

if (strcmp(L.A[i].MSSV, x) == 0) {

return i + 1; // Trả về vị trí (1-based)

}

}

return L.n + 1; // Nếu không tìm thấy, trả về n + 1

}

**Giả sử khai báo kiểu DanhSach đã được định nghĩa. Hãy viết 1 hàm trả về 1**[**danh sách sinh viên**](https://else.ctu.edu.vn/mod/quiz/view.php?id=9015)**rỗng.**

**Tên hàm: dsRong()**

DanhSach dsRong() {

DanhSach L;

L.n = 0;

return L;

}

**Tên hàm: xoaSinhVien()**

void xoaSinhVien(const char \*x, DanhSach \*pL) {

char temp[10];

int p = tim(temp, \*pL); // Gọi hàm tim với biến tạm

if (p != -1) {

xoaTai(p, pL); // Xóa sinh viên ở vị trí p

}

}

**Khai báo danh sách**

**Hãy viết khai báo kiểu DaThuc là một danh sách đặc dùng để lưu trữ 1**[**đa thức**](https://else.ctu.edu.vn/mod/quiz/view.php?id=9016)**gồm các trường:**

struct DaThuc {

DonThuc A[100]; // Mảng các đơn thức, tối đa 100 đơn thức

int so\_luong; // Số lượng đơn thức có trong đa thức

};

**Hãy viết khai báo cho kiểu dữ liệu**[**dòng văn bản**](https://else.ctu.edu.vn/mod/quiz/view.php?id=9009)**Line (trong đó Số ký tự tối đa = 80) với các thành phần được mô tả như hình trên để là một danh sách đặc các ký tự**

#define MAX\_LENGTH 80

typedef struct {

char characters[MAX\_LENGTH];

int length;

} Line;

**Giả sử người ta biểu diễn một đa giác bằng một danh sách đặc có tối đa là 100 điểm.**

**Hãy viết khai báo cho kiểu dữ liệu Polygon biểu diễn các đa giác theo mô tả trên.**

#define MAX\_POINTS 100

typedef struct {

int x;

int y;

} Point;

typedef struct {

Point points[MAX\_POINTS];

int num\_points;

} Polygon;

**Hãy viết khai báo kiểu List là một danh sách đặc dùng để lưu trữ 1**[**danh sách sinh viên**](https://else.ctu.edu.vn/mod/quiz/view.php?id=9015)**theo mô tả trên**

#define MAX\_STUDENTS 40

typedef struct {

char maSo[10];

char hoTen[50];

float diemLyThuyet;

float diemThucHanh1;

float diemThucHanh2;

} SinhVien;

typedef struct {

SinhVien A[MAX\_STUDENTS];

int n;

} List;

**Hãy viết khai báo cho kiểu dữ liệu List (trong đó Maxlength=100) với các thành phần được mô tả như hình trên để là một danh sách đặc các số thực float**

#define MAX\_LENGTH 100

typedef struct {

float A[MAX\_LENGTH];

int n;

} List;

**Dòng văn bản**

Tên hàm**: isEmpty()**

Tham số**:**L**kiểu Line**

int isEmpty(Line L) {

return (L.n == 0) ? 1 : 0;

}

Tên hàm**: printLine()**

void printLine(Line L) {

for (int i = 0; i < L.n; i++) {

putchar(L.Data[i]);

}

putchar('\n');

}

Tên hàm**: initLine()**

void initLine(Line \*pL) {

pL->n = 0;

}

Tên hàm**: appendChar()**

void appendChar(char x, Line \*pL) {

if (pL->n < MaxLength) {

pL->Data[pL->n] = x;

pL->n++;

} else {

printf("LINE IS FULL\n");

}

}

Viết hàm tìm vị trí xuất hiện đầu tiên của ký tự x trong dòng văn bản

Tên hàm**: locate()**

int locate(char x, Line L) {

for (int i = 0; i < L.n; i++) {

if (L.Data[i] == x) {

return i + 1; // Trả về vị trí (1-based index)

}

}

return L.n + 1; // Nếu không tìm thấy, trả về L.n + 1

}

Tên hàm**: removeAt()**

int removeAt(int p, Line \*pL) {

if (p < 1 || p > pL->n) {

return 0; // Vị trí không hợp lệ

}

for (int i = p - 1; i < pL->n - 1; i++) {

pL->Data[i] = pL->Data[i + 1];

}

pL->n--;

return 1; // Xóa thành công

}

Tên hàm**: erasechar()**

int erase(char ch, Line \*pL) {

int p = locate(ch, \*pL);

if (p > 0) {

return removeAt(p, pL);

}

return 0; // Ký tự không tồn tại trong dòng văn bản

}

Tên hàm**: isFull()**

int isFull(Line L) {

if (L.n == MaxLength) {

return 1; // Dòng văn bản đầy

} else {

return 0; // Dòng văn bản không đầy

}

}

Tên hàm**: countAlphabet()**

int countAlphabet(Line L) {

int count = 0;

for (int i = 0; i < L.n; i++) {

if ((L.Data[i] >= 'a' && L.Data[i] <= 'z') || (L.Data[i] >= 'A' && L.Data[i] <= 'Z')) {

count++; // Tăng biến đếm nếu ký tự là chữ cái

}

}

return count; // Trả về số lượng ký tự viết

}

**Tên hàm: countDigits()**

int countDigits(Line L) {

int count = 0;

for (int i = 0; i < L.n; i++) {

if (L.Data[i] >= '0' && L.Data[i] <= '9') {

count++;

}

}

return count;

}

Tên hàm**: appendLine()**

void appendLine(Line \*pDestination, Line Source) {

for (int i = 0; i < Source.n; i++) {

if (pDestination->n < MaxLength) {

appendChar(Source.Data[i], pDestination);

} else {

printf("LINE IS FULL\n");

break;

}

}

}

**Gọi thực thi và các phép toán của sinh viên**

Bằng cách sử dụng những phép toán đã định nghĩa trên [danh sách sinh viên](https://else.ctu.edu.vn/mod/quiz/view.php?id=9015). Hãy viết chương trình nhập và hiển thị [danh sách sinh viên](https://else.ctu.edu.vn/mod/quiz/view.php?id=9015).

#include <stdio.h>

#include <string.h>

#define MAX\_SINH\_VIEN 40

// Cấu trúc SinhVien

struct SinhVien {

char MSSV[10];

char HoTen[50];

float DiemLT, DiemTH1, DiemTH2;

};

// Cấu trúc DanhSach

struct DanhSach {

struct SinhVien A[MAX\_SINH\_VIEN];

int n;

};

// Hàm khởi tạo danh sách rỗng

struct DanhSach dsRong() {

struct DanhSach L;

L.n = 0; // Khởi tạo số lượng sinh viên

return L;

}

// Hàm tìm sinh viên theo mã số

int tim(struct DanhSach L, char \*mssv) {

for (int i = 0; i < L.n; i++) {

if (strcmp(L.A[i].MSSV, mssv) == 0) {

return i; // trả về vị trí tìm thấy

}

}

return -1; // không tìm thấy

}

// Hàm thêm sinh viên vào cuối danh sách

void chenCuoi(struct DanhSach \*L, struct SinhVien s) {

if (L->n < MAX\_SINH\_VIEN) {

L->A[L->n] = s;

L->n++;

}

}

// Hàm nhập danh sách sinh viên

struct DanhSach nhap() {

struct DanhSach L = dsRong(); // Khởi tạo danh sách rỗng

int soSinhVien;

scanf("%d", &soSinhVien);

for (int i = 0; i < soSinhVien; i++) {

struct SinhVien sv;

scanf("%s", sv.MSSV);

// Kiểm tra mã số đã tồn tại

if (tim(L, sv.MSSV) != -1) {

i--; // Giảm chỉ số để nhập lại sinh viên

continue; // Chuyển sang vòng lặp tiếp theo

}

getchar(); // Xóa ký tự newline còn lại trong buffer

fgets(sv.HoTen, sizeof(sv.HoTen), stdin);

sv.HoTen[strcspn(sv.HoTen, "\n")] = 0; // Xóa ký tự newline ở cuối chuỗi

scanf("%f %f %f", &sv.DiemLT, &sv.DiemTH1, &sv.DiemTH2);

chenCuoi(&L, sv); // Thêm sinh viên vào danh sách

}

return L; // Trả về danh sách sinh viên

}

// Hàm hiển thị danh sách sinh viên

void hienThi(struct DanhSach L) {

for (int i = 0; i < L.n; i++) {

printf("%s - %s - %.2f - %.2f - %.2f\n", L.A[i].MSSV, L.A[i].HoTen, L.A[i].DiemLT, L.A[i].DiemTH1, L.A[i].DiemTH2);

}

}

// Hàm main

int main() {

struct DanhSach L;

L = nhap(); // Nhập danh sách sinh viên

hienThi(L); // Hiển thị danh sách sinh viên

return 0;

}

Bằng cách sử dụng những phép toán đã định nghĩa trên [danh sách sinh viên](https://else.ctu.edu.vn/mod/quiz/view.php?id=9015). Hãy viết chương trình nhập và hiển thị [danh sách sinh viên](https://else.ctu.edu.vn/mod/quiz/view.php?id=9015). Sau đó liệt kê những sinh viên ĐẠT.

#include <stdio.h>

#include <string.h>

#define MAX\_SINH\_VIEN 40

// Cấu trúc SinhVien

struct SinhVien {

char MSSV[10];

char HoTen[50];

float DiemLT, DiemTH1, DiemTH2;

};

// Cấu trúc DanhSach

struct DanhSach {

struct SinhVien A[MAX\_SINH\_VIEN];

int n;

};

// Hàm khởi tạo danh sách rỗng

struct DanhSach dsRong() {

struct DanhSach L;

L.n = 0; // Khởi tạo số lượng sinh viên

return L;

}

// Hàm tìm sinh viên theo mã số

int tim(struct DanhSach L, char \*mssv) {

for (int i = 0; i < L.n; i++) {

if (strcmp(L.A[i].MSSV, mssv) == 0) {

return i; // trả về vị trí tìm thấy

}

}

return -1; // không tìm thấy

}

// Hàm thêm sinh viên vào cuối danh sách

void chenCuoi(struct DanhSach \*L, struct SinhVien s) {

if (L->n < MAX\_SINH\_VIEN) {

L->A[L->n] = s;

L->n++;

}

}

// Hàm nhập danh sách sinh viên

struct DanhSach nhap() {

struct DanhSach L = dsRong(); // Khởi tạo danh sách rỗng

int soSinhVien;

scanf("%d", &soSinhVien);

for (int i = 0; i < soSinhVien; i++) {

struct SinhVien sv;

scanf("%s", sv.MSSV);

// Kiểm tra mã số đã tồn tại

if (tim(L, sv.MSSV) != -1) {

i--; // Giảm chỉ số để nhập lại sinh viên

continue; // Chuyển sang vòng lặp tiếp theo

}

getchar(); // Xóa ký tự newline còn lại trong buffer

fgets(sv.HoTen, sizeof(sv.HoTen), stdin);

sv.HoTen[strcspn(sv.HoTen, "\n")] = 0; // Xóa ký tự newline ở cuối chuỗi

scanf("%f %f %f", &sv.DiemLT, &sv.DiemTH1, &sv.DiemTH2);

chenCuoi(&L, sv); // Thêm sinh viên vào danh sách

}

return L; // Trả về danh sách sinh viên

}

// Hàm hiển thị danh sách sinh viên

void hienThi(struct DanhSach L) {

for (int i = 0; i < L.n; i++) {

printf("%s - %s - %.2f - %.2f - %.2f\n", L.A[i].MSSV, L.A[i].HoTen, L.A[i].DiemLT, L.A[i].DiemTH1, L.A[i].DiemTH2);

}

}

// Hàm liệt kê sinh viên đạt

void lietKeDat(struct DanhSach L) {

printf("Sinh vien DAT\n");

for (int i = 0; i < L.n; i++) {

float tongDiem = L.A[i].DiemLT + L.A[i].DiemTH1 + L.A[i].DiemTH2;

if (tongDiem >= 4.0) {

printf("%s - %s - %.2f - %.2f - %.2f\n", L.A[i].MSSV, L.A[i].HoTen, L.A[i].DiemLT, L.A[i].DiemTH1, L.A[i].DiemTH2);

}

}

}

// Hàm main

int main() {

struct DanhSach L;

L = nhap(); // Nhập danh sách sinh viên

hienThi(L); // Hiển thị danh sách sinh viên

lietKeDat(L); // Liệt kê sinh viên đạt

return 0;

}

Bằng cách sử dụng những phép toán đã định nghĩa trên [danh sách sinh viên](https://else.ctu.edu.vn/mod/quiz/view.php?id=9015). Hãy viết chương trình nhập và hiển thị [danh sách sinh viên](https://else.ctu.edu.vn/mod/quiz/view.php?id=9015). Sau đó liệt kê những sinh viên KHÔNG ĐẠT.

#include <stdio.h>

#include <string.h>

#define MAX\_SINH\_VIEN 40

// Cấu trúc SinhVien

struct SinhVien {

char MSSV[10];

char HoTen[50];

float DiemLT, DiemTH1, DiemTH2;

};

// Cấu trúc DanhSach

struct DanhSach {

struct SinhVien A[MAX\_SINH\_VIEN];

int n;

};

// Hàm khởi tạo danh sách rỗng

struct DanhSach dsRong() {

struct DanhSach L;

L.n = 0; // Khởi tạo số lượng sinh viên

return L;

}

// Hàm tìm sinh viên theo mã số

int tim(struct DanhSach L, char \*mssv) {

for (int i = 0; i < L.n; i++) {

if (strcmp(L.A[i].MSSV, mssv) == 0) {

return i; // trả về vị trí tìm thấy

}

}

return -1; // không tìm thấy

}

// Hàm thêm sinh viên vào cuối danh sách

void chenCuoi(struct DanhSach \*L, struct SinhVien s) {

if (L->n < MAX\_SINH\_VIEN) {

L->A[L->n] = s;

L->n++;

}

}

// Hàm nhập danh sách sinh viên

struct DanhSach nhap() {

struct DanhSach L = dsRong(); // Khởi tạo danh sách rỗng

int soSinhVien;

scanf("%d", &soSinhVien);

for (int i = 0; i < soSinhVien; i++) {

struct SinhVien sv;

scanf("%s", sv.MSSV);

// Kiểm tra mã số đã tồn tại

if (tim(L, sv.MSSV) != -1) {

i--; // Giảm chỉ số để nhập lại sinh viên

continue; // Chuyển sang vòng lặp tiếp theo

}

getchar(); // Xóa ký tự newline còn lại trong buffer

fgets(sv.HoTen, sizeof(sv.HoTen), stdin);

sv.HoTen[strcspn(sv.HoTen, "\n")] = 0; // Xóa ký tự newline ở cuối chuỗi

scanf("%f %f %f", &sv.DiemLT, &sv.DiemTH1, &sv.DiemTH2);

chenCuoi(&L, sv); // Thêm sinh viên vào danh sách

}

return L; // Trả về danh sách sinh viên

}

// Hàm hiển thị danh sách sinh viên

void hienThi(struct DanhSach L) {

for (int i = 0; i < L.n; i++) {

printf("%s - %s - %.2f - %.2f - %.2f\n", L.A[i].MSSV, L.A[i].HoTen, L.A[i].DiemLT, L.A[i].DiemTH1, L.A[i].DiemTH2);

}

}

// Hàm liệt kê sinh viên không đạt

void lietKeKhongDat(struct DanhSach L) {

printf("Sinh vien KHONG DAT\n");

for (int i = 0; i < L.n; i++) {

float tongDiem = L.A[i].DiemLT + L.A[i].DiemTH1 + L.A[i].DiemTH2;

if (tongDiem < 4.0) {

printf("%s - %s - %.2f - %.2f - %.2f\n", L.A[i].MSSV, L.A[i].HoTen, L.A[i].DiemLT, L.A[i].DiemTH1, L.A[i].DiemTH2);

}

}

}

// Hàm main

int main() {

struct DanhSach L;

L = nhap(); // Nhập danh sách sinh viên

hienThi(L); // Hiển thị danh sách sinh viên

lietKeKhongDat(L); // Liệt kê sinh viên không đạt

return 0;

}

Bằng cách sử dụng những phép toán đã định nghĩa trên [danh sách sinh viên](https://else.ctu.edu.vn/mod/quiz/view.php?id=9015). Hãy viết chương trình nhập và hiển thị [danh sách sinh viên](https://else.ctu.edu.vn/mod/quiz/view.php?id=9015). Sau đó nhập một mã số sinh viên từ bàn phím, nếu mã số tồn tại thì xóa sinh viên có mã số đó.

#include <stdio.h>

#include <string.h>

#define MAX\_SINH\_VIEN 40

// Cấu trúc SinhVien

struct SinhVien {

char MSSV[10];

char HoTen[50];

float DiemLT, DiemTH1, DiemTH2;

};

// Cấu trúc DanhSach

struct DanhSach {

struct SinhVien A[MAX\_SINH\_VIEN];

int n;

};

// Hàm khởi tạo danh sách rỗng

struct DanhSach dsRong() {

struct DanhSach L;

L.n = 0; // Khởi tạo số lượng sinh viên

return L;

}

// Hàm tìm sinh viên theo mã số

int tim(struct DanhSach L, char \*mssv) {

for (int i = 0; i < L.n; i++) {

if (strcmp(L.A[i].MSSV, mssv) == 0) {

return i; // trả về vị trí tìm thấy

}

}

return -1; // không tìm thấy

}

// Hàm thêm sinh viên vào cuối danh sách

void chenCuoi(struct DanhSach \*L, struct SinhVien s) {

if (L->n < MAX\_SINH\_VIEN) {

L->A[L->n] = s;

L->n++;

}

}

// Hàm xóa sinh viên theo mã số

void xoa(struct DanhSach \*L, char \*mssv) {

int viTri = tim(\*L, mssv);

if (viTri != -1) {

for (int i = viTri; i < L->n - 1; i++) {

L->A[i] = L->A[i + 1]; // Dịch chuyển phần tử

}

L->n--; // Giảm số lượng sinh viên

} else {

printf("Khong tim thay sinh vien %s\n", mssv);

}

}

// Hàm nhập danh sách sinh viên

struct DanhSach nhap() {

struct DanhSach L = dsRong(); // Khởi tạo danh sách rỗng

int soSinhVien;

scanf("%d", &soSinhVien);

for (int i = 0; i < soSinhVien; i++) {

struct SinhVien sv;

scanf("%s", sv.MSSV);

// Kiểm tra mã số đã tồn tại

if (tim(L, sv.MSSV) != -1) {

i--; // Giảm chỉ số để nhập lại sinh viên

continue; // Chuyển sang vòng lặp tiếp theo

}

getchar(); // Xóa ký tự newline còn lại trong buffer

fgets(sv.HoTen, sizeof(sv.HoTen), stdin);

sv.HoTen[strcspn(sv.HoTen, "\n")] = 0; // Xóa ký tự newline ở cuối chuỗi

scanf("%f %f %f", &sv.DiemLT, &sv.DiemTH1, &sv.DiemTH2);

chenCuoi(&L, sv); // Thêm sinh viên vào danh sách

}

return L; // Trả về danh sách sinh viên

}

// Hàm hiển thị danh sách sinh viên

void hienThi(struct DanhSach L) {

for (int i = 0; i < L.n; i++) {

printf("%s - %s - %.2f - %.2f - %.2f\n", L.A[i].MSSV, L.A[i].HoTen, L.A[i].DiemLT, L.A[i].DiemTH1, L.A[i].DiemTH2);

}

}

// Hàm main

int main() {

struct DanhSach L;

L = nhap(); // Nhập danh sách sinh viên

hienThi(L); // Hiển thị danh sách sinh viên

char mssv[10];

scanf("%s", mssv);

// Tìm kiếm và xóa sinh viên

int viTri = tim(L, mssv);

if (viTri != -1) {

printf("Tim thay sinh vien %s. Thong tin sinh vien:\n", mssv);

printf("%s - %s - %.2f - %.2f - %.2f\n", L.A[viTri].MSSV, L.A[viTri].HoTen, L.A[viTri].DiemLT, L.A[viTri].DiemTH1, L.A[viTri].DiemTH2);

xoa(&L, mssv); // Xóa sinh viên

} else {

printf("Khong tim thay sinh vien %s\n", mssv);

}

return 0;

}

Danh sách liên kết

Hãy viết khai báo cho kiểu dữ liệu **List** với các thành phần được mô tả như hình trên để là một danh sách liên kết đơn các số nguyên

typedef struct Node {

int Element;

struct Node\* Next;

} Node;

typedef Node\* Position;

typedef Node\* List;

Tên hàm: **makenullList()**

void makenullList(List \*pL) {

\*pL = (List)malloc(sizeof(struct Node)); // Cấp phát bộ nhớ cho nút đầu tiên

(\*pL)->Next = NULL; // Trường Next chỉ tới NULL

}

Tên hàm (function name): **member()**

int member(int x, List L) {

for (Position p = L->Next; p; p = p->Next)

if (p->Element == x) return 1;

return 0;

}

Tên hàm (function name): **sort**()

void sort(List\* pL) {

for (Position p = (\*pL)->Next; p; p = p->Next) {

for (Position q = p->Next; q; q = q->Next) {

if (p->Element > q->Element) {

int temp = p->Element;

p->Element = q->Element;

q->Element = temp;

}

}

}

}

**Tên hàm (function name): unionSet()**

List unionSet(List L1, List L2) {

List result;

makenullList(&result);

for (Position p = L1->Next; p; p = p->Next) append(p->Element, &result);

for (Position p = L2->Next; p; p = p->Next)

if (!member(p->Element, result)) append(p->Element, &result);

return result;

}

Tên hàm: **deleteList**()

void deleteList(Position p, List \*pL) {

if (p != NULL && p->Next != NULL) {

Position temp = p->Next; // Lưu vị trí của nút cần xóa

p->Next = p->Next->Next; // Cập nhật liên kết để bỏ qua nút cần xóa

free(temp); // Giải phóng bộ nhớ

}

}

Tên hàm: removeAll()

void removeAll(int x, List \*pL) {

for (Position i = \*pL; i->Next; )

if (i == locate(x, \*pL))

deleteList(i, pL);

else

i = i->Next;

}

Xóa phần tử đầu tiên

void deleteHead(List \*pL) {

if (pL->Head != NULL) { // Kiểm tra xem danh sách có nút nào không

Position temp = pL->Head; // Lưu vị trí của nút Head

pL->Head = pL->Head->Next; // Cập nhật Head để trỏ đến nút tiếp theo

free(temp); // Giải phóng bộ nhớ cho nút A

}

}

Tên hàm (function name): **intersection**()

List intersection(List L1, List L2) {

List result;

makenullList(&result);

for (Position p = L1->Next; p != NULL; p = p->Next) {

if (member(p->Element, L2)) append(p->Element, &result);

}

return result;

}

Tên hàm (function name): **copyEvenNumbers()**

void copyEvenNumbers(List L1, List \*pL2) {

makenullList(pL2);

for (Position p = L1->Next; p != NULL; p = p->Next) {

if (p->Element % 2 == 0) append(p->Element, pL2);

}

}

Tên hàm: **append**()

void append(int x, List \*pL) {

Position newNode = (Position)malloc(sizeof(struct Node));

newNode->Element = x;

newNode->Next = NULL;

Position p = \*pL;

while (p->Next) p = p->Next;

p->Next = newNode;

}

Tên hàm: **insertList**()

void insertList(int x, Position p, List \*pL) {

Position newNode = (Position)malloc(sizeof(struct Node));

newNode->Element = x;

newNode->Next = p->Next;

p->Next = newNode;

}

Tên hàm: **locate**()

Position locate(ElementType x, List L) {

Position p = L;

while (p->Next != NULL && p->Next->Element != x) {

p = p->Next;

}

return p;

}

Tên hàm (function name): **normalize**()

void normalize(List \*pL) {

for (Position p = (\*pL)->Next; p; p = p->Next) {

Position prev = p;

for (Position q = p->Next; q; q = prev->Next) {

if (p->Element == q->Element) {

deleteList(prev, pL);

} else {

prev = q;

}

}

}

}

Tên hàm (function name): **printOddNumbers()**

void printOddNumbers(List L) {

for (Position p = L->Next; p; p = p->Next)

if (p->Element % 2 != 0)

printf("%d ", p->Element);

}

Tên hàm (function name): **getAvg()**

float getAvg(List L) {

if (L->Next == NULL) return -10000.0f;

float sum = 0.0f;

int count = 0;

for (Position p = L->Next; p != NULL; p = p->Next) {

sum += p->Element;

count++;

}

return sum / count;

}

Tên hàm: **readList()**

void readList(List \*pL) {

int n, x;

makenullList(pL);

scanf("%d", &n);

for (Position p = \*pL; n--; p = p->Next) {

scanf("%d", &x);

p->Next = (Position)malloc(sizeof(struct Node));

p->Next->Element = x;

p->Next->Next = NULL;

}

}

Tên hàm (function name): **readSet()**

List readSet() {

List L;

makenullList(&L);

int n, x;

scanf("%d", &n);

while (n--) {

scanf("%d", &x);

if (!member(x, L)) addFirst(x, &L);

}

return L;

}

Tên hàm (function name): **difference**()

List difference(List L1, List L2) {

List result;

makenullList(&result);

for (Position p = L1->Next; p; p = p->Next)

if (!member(p->Element, L2))

append(p->Element, &result);

return result;

}

Tên hàm (function name): **erase()**

void erase(int x, List \*pL) {

Position p = locate(x, \*pL);

if (p->Next) {

Position temp = p->Next;

p->Next = temp->Next;

free(temp);

} else {

printf("Not found %d\n", x);

}

}

Tên hàm: **addFirst**()

void addFirst(int x, List\* pL) {

Position newNode = (Position)malloc(sizeof(struct Node));

newNode->Element = x;

newNode->Next = (\*pL)->Next;

(\*pL)->Next = newNode;

}

**Bằng cách sử dụng những phép toán chuẩn trên danh sách, hãy viết chương trình cho phép liệt kê các số chẵn và tính trung bình cộng các phần tử là số chẵn trong danh sách**

#include <stdio.h>

#include <stdlib.h>

typedef struct Node {

int Element;

struct Node\* Next;

} Node;

typedef Node\* List;

typedef Node\* Position;

// Khởi tạo danh sách rỗng

void makenullList(List \*pL) {

\*pL = (List)malloc(sizeof(Node));

(\*pL)->Next = NULL;

}

// Thêm phần tử vào cuối danh sách

void append(int x, List \*pL) {

Position newNode = (Position)malloc(sizeof(Node));

newNode->Element = x;

newNode->Next = NULL;

Position p = \*pL;

while (p->Next != NULL) {

p = p->Next;

}

p->Next = newNode;

}

// Hiển thị danh sách

void displayList(List L) {

Position p = L->Next;

while (p != NULL) {

printf("%d ", p->Element);

p = p->Next;

}

printf("\n");

}

// Tính trung bình cộng các số chẵn

float getAvgEven(List L) {

float sum = 0;

int count = 0;

Position p = L->Next;

while (p != NULL) {

if (p->Element % 2 == 0) {

sum += p->Element;

count++;

}

p = p->Next;

}

return (count == 0) ? -10000.0f : sum / count;

}

// Chép các số chẵn sang danh sách kết quả

void copyEvenNumbers(List L1, List \*pL2) {

makenullList(pL2);

Position p = L1->Next;

while (p != NULL) {

if (p->Element % 2 == 0) {

append(p->Element, pL2);

}

p = p->Next;

}

}

int main() {

List L, evenList;

int n, x;

scanf("%d", &n);

makenullList(&L);

for (int i = 0; i < n; i++) {

scanf("%d", &x);

append(x, &L);

}

displayList(L);

copyEvenNumbers(L, &evenList);

displayList(evenList);

float avg = getAvgEven(L);

printf("%.3f\n", avg);

return 0;

}

**Bằng cách sử dụng những phép toán chuẩn trên danh sách, hãy viết chương trình cho phép sắp xếp danh sách tăng dần**

#include <stdio.h>

#include <stdlib.h>

typedef struct Node {

int Element;

struct Node\* Next;

} Node;

typedef Node\* List;

typedef Node\* Position;

// Khởi tạo danh sách rỗng

void makenullList(List \*pL) {

\*pL = (List)malloc(sizeof(Node));

(\*pL)->Next = NULL;

}

// Thêm phần tử vào cuối danh sách

void append(int x, List \*pL) {

Position newNode = (Position)malloc(sizeof(Node));

newNode->Element = x;

newNode->Next = NULL;

Position p = \*pL;

while (p->Next != NULL) {

p = p->Next;

}

p->Next = newNode;

}

// Hiển thị danh sách

void displayList(List L) {

Position p = L->Next;

while (p != NULL) {

printf("%d ", p->Element);

p = p->Next;

}

printf("\n");

}

// Sắp xếp danh sách tăng dần

void sortList(List L) {

if (L->Next == NULL) return; // Danh sách rỗng

Position p, q;

int temp;

for (p = L->Next; p->Next != NULL; p = p->Next) {

for (q = p->Next; q != NULL; q = q->Next) {

if (p->Element > q->Element) {

// Đổi chỗ giá trị của p và q

temp = p->Element;

p->Element = q->Element;

q->Element = temp;

}

}

}

}

int main() {

List L;

int n, x;

scanf("%d", &n);

makenullList(&L);

for (int i = 0; i < n; i++) {

scanf("%d", &x);

append(x, &L);

}

displayList(L);

sortList(L);

displayList(L);

return 0;

}

**Bằng cách sử dụng những phép toán chuẩn trên danh sách, hãy viết chương trình cho phép nhập và hiển thị một danh sách số nguyên từ bàn phím**

#include <stdio.h>

#include <stdlib.h>

typedef struct Node {

int Element;

struct Node\* Next;

} Node;

typedef Node\* List;

typedef Node\* Position;

// Hàm khởi tạo danh sách rỗng

void makenullList(List \*pL) {

\*pL = (List)malloc(sizeof(Node));

(\*pL)->Next = NULL;

}

// Hàm thêm phần tử vào cuối danh sách

void append(int x, List \*pL) {

Position newNode = (Position)malloc(sizeof(Node));

newNode->Element = x;

newNode->Next = NULL;

Position p = \*pL;

while (p->Next != NULL) {

p = p->Next;

}

p->Next = newNode;

}

// Hàm hiển thị danh sách

void displayList(List L) {

Position p = L->Next; // Bỏ qua phần tử đầu tiên

while (p != NULL) {

printf("%d ", p->Element);

p = p->Next;

}

printf("\n");

}

int main() {

List L;

int n, x;

scanf("%d", &n);

makenullList(&L);

for (int i = 0; i < n; i++) {

scanf("%d", &x);

append(x, &L);

}

displayList(L);

return 0;

}

**Bằng cách sử dụng những phép toán trên tập hợp, hãy viết chương trình cho phép tìm tập hợp giao của 2 danh sách biểu diễn tập hợp**

#include <stdio.h>

#include <stdlib.h>

typedef struct Node{

int Element;

struct Node\* Next;

}Node;

typedef Node\* pos;

typedef Node\* List;

void makenullList(List \*pL){

\*pL = (List)malloc(sizeof(struct Node));

(\*pL)->Next = NULL;

}

void addFirst(int x, List \*pL){

pos n = (pos)malloc(sizeof(struct Node));

n->Element = x;

n->Next = (\*pL)->Next;

(\*pL)->Next = n;

}

int member(int x, List L){

pos p;

for(p = L->Next; p; p = p->Next)

if(p->Element == x) return 1;

return 0;

}

List readSet(){

List L;

makenullList(&L);

int n, x;

scanf("%d",&n);

while(n--){

scanf("%d",&x);

if(!member(x,L)) addFirst(x,&L);

}

return L;

}

void printList(List L){

pos p;

for(p = L->Next; p; p = p->Next)

printf("%d ",p->Element);

printf("\n");

}

void append(int x, List \*pL){

pos n = (pos)malloc(sizeof(struct Node));

n->Element = x;

n->Next = NULL;

pos p = \*pL;

while(p->Next)u

p = p->Next;

p->Next = n;

}

List intersection(List L1, List L2){

List L;

makenullList(&L);

pos p;

for(p = L1->Next; p; p = p->Next){

if(member(p->Element,L2)) append(p->Element,&L);

}

return L;

}

int main(){

List L1 = readSet();

List L2 = readSet();

printList(L1);

printList(L2);

List L = intersection(L1,L2);

printList(L);

return 0;

}

**Bằng cách sử dụng những phép toán chuẩn trên danh sách, hãy viết chương trình cho phép xóa phần tử x xuất hiện đầu tiên trong danh sách**

#include <stdio.h>

#include <stdlib.h>

typedef struct Node{

int Element;

struct Node\* Next;

}Node;

typedef Node\* Pos;

typedef Node\* List;

void makenullList(List \*pL){

\*pL = (Pos)malloc(sizeof(struct Node));

(\*pL)->Next = 0;

}

void append(int x, List \*pL){

Pos n = (Pos)malloc(sizeof(struct Node));

n->Element = x;

n->Next = NULL;

Pos p = \*pL;

while(p->Next)

p = p->Next;

p->Next = n;

}

List read(){

List L;

makenullList(&L);

int n, x;

scanf("%d",&n);

while(n--){

scanf("%d",&x);

append(x,&L);

}

return L;

}

void printList(List L){

Pos p;

for(p = L->Next; p; p = p->Next)

printf("%d ",p->Element);

printf("\n");

}

Pos locate(int x, List L){

Pos p = L;

while(p->Next!=NULL && p->Next->Element != x)

p = p->Next;

return p;

}

void deleteList(Pos p, List \*pL){

if(p!=NULL && p->Next!=NULL){

Pos tmp = p->Next;

p->Next = p->Next->Next;

free(tmp);

}

}

void removeAll(int x, List \*pL){

Pos p;

for(p = \*pL; p->Next;){

if(p==locate(x,\*pL)) deleteList(p,pL);

else p = p->Next;

}

}

int main(){

List L = read();

printList(L);

int x;

scanf("%d",&x);

removeAll(x,&L);

printList(L);

return 0;

}

Bằng cách sử dụng những phép toán chuẩn trên danh sách, hãy viết chương trình cho phép tính trung bình cộng các phần tử trong danh sách

#include <stdio.h>

#include <stdlib.h>

typedef struct Node {

int Element;

struct Node\* Next;

} Node;

typedef Node\* List;

// Hàm khởi tạo danh sách rỗng

void makenullList(List \*pL) {

\*pL = (List)malloc(sizeof(Node));

(\*pL)->Next = NULL;

}

// Hàm thêm phần tử vào danh sách

void append(int x, List \*pL) {

Node\* newNode = (Node\*)malloc(sizeof(Node));

newNode->Element = x;

newNode->Next = NULL;

Node\* p = \*pL;

while (p->Next != NULL) {

p = p->Next;

}

p->Next = newNode; // Thêm vào cuối danh sách

}

// Hàm hiển thị danh sách

void displayList(List L) {

Node\* p = L->Next; // Bỏ qua phần tử đầu tiên

while (p != NULL) {

printf("%d ", p->Element);

p = p->Next;

}

printf("\n");

}

// Hàm tính trung bình cộng các phần tử trong danh sách

double calculateAverage(List L) {

if (L->Next == NULL) return 0.0; // Nếu danh sách rỗng

Node\* p = L->Next;

int sum = 0, count = 0;

while (p != NULL) {

sum += p->Element;

count++;

p = p->Next;

}

return (double)sum / count; // Tính trung bình cộng

}

int main() {

List L;

int n;

// Nhập danh sách

scanf("%d", &n);

makenullList(&L);

for (int i = 0; i < n; i++) {

int element;

scanf("%d", &element);

append(element, &L);

}

// Hiển thị danh sách

displayList(L);

// Tính và hiển thị trung bình cộng

double average = calculateAverage(L);

printf("%.3f\n", average);

// Giải phóng bộ nhớ

Node\* current = L;

Node\* nextNode;

while (current != NULL) {

nextNode = current->Next;

free(current);

current = nextNode;

}

return 0;

}

Bằng cách sử dụng những phép toán chuẩn trên danh sách, hãy viết chương trình cho phép liệt kê các số lẻ, số chẵn trong danh sách

#include <stdio.h>

#include <stdlib.h>

typedef struct Node{

int Element;

struct Node\* Next;

}Node;

typedef Node\* List;

typedef Node\* pos;

void makenullList(List \*pL){

\*pL = (List)malloc(sizeof(struct Node));

(\*pL)->Next = NULL;

}

void append(int x, List \*pL){

pos n = (pos)malloc(sizeof(struct Node));

n->Element = x;

n->Next = NULL;

pos p = \*pL;

while(p->Next)

p = p->Next;

p->Next = n;

}

List readSet(){

List L;

makenullList(&L);

int n, x;

scanf("%d",&n);

while(n--){

scanf("%d",&x);

append(x,&L);

}

return L;

}

void print(List L){

pos p;

for(p = L->Next; p; p = p->Next)

printf("%d ",p->Element);

printf("\n");

for(p = L->Next; p; p = p->Next)

if(p->Element % 2 != 0) printf("%d ",p->Element);

printf("\n");

for(p = L->Next; p; p = p->Next)

if(p->Element % 2 == 0) printf("%d ",p->Element);

}

int main() {

List L = readSet();

print(L);

return 0;

}

Bằng cách sử dụng những phép toán trên tập hợp, hãy viết chương trình cho phéptìm tập hợp hợp của 2 danh sách biểu diễn tập hợp

#include <stdio.h>

#include <stdlib.h>

typedef struct Node{

int Element;

struct Node\* Next;

}Node;

typedef Node\* Pos;

typedef Node\* List;

void makenullList(List \*pL){

\*pL = (Pos)malloc(sizeof(struct Node));

(\*pL)->Next = 0;

}

void append(int x, List \*pL){

Pos n = (Pos)malloc(sizeof(struct Node));

n->Element = x;

n->Next = NULL;

Pos p = \*pL;

while(p->Next)

p = p->Next;

p->Next = n;

}

void addFirst(int x, List \*pL){

Pos n = (Pos)malloc(sizeof(struct Node));

n->Element = x;

n->Next = (\*pL)->Next;

(\*pL)->Next = n;

}

int member(int x, List L){

Pos p;

for(p = L->Next; p; p = p->Next)

if(p->Element == x) return 1;

return 0;

}

List read(){

List L;

makenullList(&L);

int n, x;

scanf("%d",&n);

while(n--){

scanf("%d",&x);

if(!member(x,L))

addFirst(x,&L);

}

return L;

}

void printList(List L){

Pos p;

for(p = L->Next; p; p = p->Next)

printf("%d ",p->Element);

printf("\n");

}

List unionSet(List L1, List L2){

List L;

makenullList(&L);

Pos p, q;

for(p = L1->Next; p; p = p->Next)

append(p->Element,&L);

for(q = L2->Next; q; q = q->Next)

if(!member(q->Element,L)) append(q->Element,&L);

return L;

}

int main(){

List L1 = read();

List L2 = read();

printList(L1);

printList(L2);

List L = unionSet(L1, L2);

printList(L);

return 0;

}

Ngăn Xếp

Tên hàm: **giatriDinh**()

int giatriDinh(NganXep S){

return S.DuLieu[S.Dinh];

}

**Khai báo NganXep**

#define SoPhanTu 100

typedef struct{

int DuLieu[SoPhanTu];

int Dinh;

}NganXep;

Tên hàm: **ktDay**()

int ktDay(NganXep S){

return S.Dinh ==0;

}

Tên hàm: **ktRong**()

int ktRong(NganXep S){

return S.Dinh == SoPhanTu;

}

Tên hàm: **hienthi**()

void hienthi(NganXep \*pS){

while(!ktRong(\*pS)){

printf("%d ",giatriDinh(\*pS));

xoa(pS);

}

printf("\n");

}

Tên hàm: **them**()

void them(int X, NganXep \*pS){

if(!ktDay(\*pS)){

pS->Dinh--;

pS->DuLieu[pS->Dinh] = X;

}

}

Tên hàm: **xoa**()

void xoa(NganXep \*pS){

if (!ktRong(\*pS))

pS->Dinh++;

}

Tên hàm: **khoitao**()

void khoitao(NganXep \*pS){

pS->Dinh = SoPhanTu;

}

**ỨNG DỤNG NGĂN XẾP**

Tên hàm: **doiNhiPhan**()

void doiNhiPhan(int n, NganXep \*pS) {

khoitao(pS); // Khởi tạo ngăn xếp

// Chuyển đổi số nguyên dương n về dạng nhị phân

while (n > 0) {

int du = n % 2; // Lấy phần dư khi chia n cho 2

them(du, pS); // Thêm phần dư vào ngăn xếp

n = n / 2; // Giảm n đi một nửa

}

}

void hienthi(NganXep \*pS) {

while (!ktRong(\*pS)) {

int giaTri = giatriDinh(\*pS);

printf("%d ", giaTri);

xoa(pS);

}

}

Hàm **inThaplucphan()**

void inThaplucphan(int n) {

Stack s;

makenullStack(&s); // Khởi tạo ngăn xếp

// Mảng chứa các ký tự thập lục phân

char hexDigits[] = "0123456789ABCDEF";

// Chuyển đổi số nguyên n về dạng thập lục phân

while (n > 0) {

int du = n % 16; // Lấy phần dư khi chia n cho 16

push(hexDigits[du], &s); // Thêm ký tự tương ứng vào ngăn xếp

n = n / 16; // Giảm n đi một phần mười sáu

}

// In kết quả từ ngăn xếp

while (!emptyStack(s)) {

char c = top(s); // Lấy giá trị tại đỉnh ngăn xếp

pop(&s); // Xóa phần tử tại đỉnh ngăn xếp

printf("%c", c); // In ký tự

}

printf("\n"); // Xuống dòng sau khi in xong

}

Hàm **ktChuoi()**

#define MAX\_LENGTH 100 // Độ dài tối đa của chuỗi

// Giả sử kiểu dữ liệu Stack và các hàm liên quan đã được định nghĩa ở đây

int ktChuoi() {

Stack S;

makenullStack(&S); // Khởi tạo ngăn xếp

char str[MAX\_LENGTH];

fgets(str, MAX\_LENGTH, stdin); // Nhập chuỗi

// Loại bỏ ký tự newline nếu có

str[strcspn(str, "\n")] = '\0';

// Duyệt qua từng ký tự trong chuỗi

for (int i = 0; i < strlen(str); i++) {

char ch = str[i];

if (ch == '(') {

push(ch, &S); // Nếu là '(', đưa vào ngăn xếp

} else if (ch == ')') {

if (emptyStack(S)) {

return 0; // Ngăn xếp rỗng, chuỗi không đúng

}

pop(&S); // Xóa phần tử trên đỉnh ngăn xếp

}

}

// Kiểm tra nếu ngăn xếp rỗng

return emptyStack(S) ? 1 : 0; // 1 nếu đúng, 0 nếu không đúng

}

Hàm **tinhGiatri()**

#define MAX\_LENGTH 100 // Độ dài tối đa của chuỗi

// Giả sử kiểu dữ liệu Stack và các hàm liên quan đã được định nghĩa ở đây

double tinhGiatri(char \*expression) {

Stack S;

makenullStack(&S); // Khởi tạo ngăn xếp

for (int i = 0; i < strlen(expression); i++) {

char ch = expression[i];

if (isdigit(ch)) {

// Nếu là ký tự số, chuyển thành số thực và đưa vào ngăn xếp

double num = ch - '0';

push(num, &S);

} else if (ch == ' ') {

// Bỏ qua ký tự khoảng trắng

continue;

} else {

// Nếu là ký tự phép toán, lấy hai phần tử trên đỉnh ngăn xếp ra

double val2 = top(S); pop(&S);

double val1 = top(S); pop(&S);

double result;

// Thực hiện phép toán tương ứng

switch (ch) {

case '+': result = val1 + val2; break;

case '-': result = val1 - val2; break;

case '\*': result = val1 \* val2; break;

case '/': result = val1 / val2; break;

default: printf("Phép toán không hợp lệ\n"); return 0;

}

// Đưa kết quả phép toán vào ngăn xếp

push(result, &S);

}

}

// Giá trị cuối cùng trên đỉnh ngăn xếp là kết quả của biểu thức hậu tố

return top(S);

}

Hàm **chuyenHauto()**

#define MAX\_LENGTH 100 // Độ dài tối đa của chuỗi

// Giả sử kiểu dữ liệu Stack và các hàm liên quan đã được định nghĩa ở đây

void chuyenHauto(char \*trungto, char \*hauto) {

Stack S;

makenullStack(&S); // Khởi tạo ngăn xếp

int j = 0; // Chỉ số cho chuỗi hauto

for (int i = 0; i < strlen(trungto); i++) {

char c = trungto[i];

if (isalnum(c)) {

// Nếu là toán hạng (số hoặc chữ cái), đưa vào chuỗi hauto

hauto[j++] = c;

} else if (c == '(') {

// Nếu là dấu '(', đưa vào ngăn xếp

push(c, &S);

} else if (c == ')') {

// Nếu là dấu ')', lấy các toán tử ra khỏi ngăn xếp cho đến khi gặp '('

while (!emptyStack(S) && top(S) != '(') {

hauto[j++] = top(S);

pop(&S);

}

pop(&S); // Xóa dấu '(' ra khỏi ngăn xếp

} else if (c == '+' || c == '-' || c == '\*' || c == '/') {

// Nếu là toán tử, xử lý theo mức ưu tiên

while (!emptyStack(S) && top(S) != '(' && mucUutien(c) <= mucUutien(top(S))) {

hauto[j++] = top(S);

pop(&S);

}

push(c, &S);

}

}

// Đưa tất cả các toán tử còn lại trong ngăn xếp vào chuỗi hauto

while (!emptyStack(S)) {

hauto[j++] = top(S);

pop(&S);

}

hauto[j] = '\0'; // Kết thúc chuỗi hauto

}

// int main() {

// char trungto[MAX\_LENGTH] = "(3+4)/2";

// char hauto[MAX\_LENGTH];

// chuyenHauto(trungto, hauto);

// printf("Biểu thức hậu tố: %s\n", hauto);

// return 0;

//}

Hàng Đợi

**Tính số đơn vị thời gian cần thiết của Minh để hoàn thành n công việc được giao.**

#include <stdio.h>

#include <stdlib.h>

#define MAX\_N 100

int main() {

int N;

scanf("%d", &N);

int tasks[N]; // Danh sách các công việc được giao

int optimal[N]; // Danh sách các công việc tối ưu

int time = 0; // Thời gian cần thiết

// Nhập danh sách công việc được giao

for (int i = 0; i < N; i++) {

scanf("%d", &tasks[i]);

}

// Nhập danh sách công việc tối ưu

for (int i = 0; i < N; i++) {

scanf("%d", &optimal[i]);

}

int optimalIndex = 0; // Chỉ số cho danh sách tối ưu

// Sử dụng vòng lặp để xử lý các công việc

for (int i = 0; i < N;) {

// Nếu công việc hiện tại khớp với công việc tối ưu

if (tasks[i] == optimal[optimalIndex]) {

// Thực hiện công việc

optimalIndex++;

time++; // Tăng thời gian

i++; // Chuyển sang công việc tiếp theo trong danh sách giao

} else {

// Nếu không khớp, chuyển công việc vào cuối

int temp = tasks[i];

for (int j = i; j < N - 1; j++) {

tasks[j] = tasks[j + 1]; // Di chuyển công việc

}

tasks[N - 1] = temp; // Đưa công việc vào cuối danh sách

time++; // Tăng thời gian

}

// Nếu đã thực hiện xong tất cả công việc tối ưu

if (optimalIndex >= N) {

break;

}

// Nếu danh sách công việc đã trống

if (i >= N) {

i = 0; // Reset chỉ số về đầu danh sách

}

}

printf("%d\n", time);

return 0;

}

**2**. Minh có một hàng đợi Q và cô ấy muốn thực hiện N thao tác, mỗi thao tác là một trong 2 dạng sau:

**E x**: thêm x vào hàng đợi và in ra số lượng phần tử của hàng đợi

**D**: xóa phần tử đầu hàng, đồng thời in phần tử đã xóa và số lượng phần tử của hàng đợi sau khi xóa, 2 giá trị này cách nhau khoảng trắng. Nếu hàng đợi rỗng, in -1 ở chỗ phần tử đã xóa.

Hãy giúp Minh Minh thực hiện các thao tác trên.

#include <stdio.h>

#include <stdlib.h>

#define MAX\_SIZE 100

typedef struct {

int data[MAX\_SIZE];

int front;

int rear;

} Queue;

// Hàm khởi tạo hàng đợi

void initializeQueue(Queue \*q) {

q->front = 0;

q->rear = 0;

}

// Hàm kiểm tra hàng đợi rỗng

int isEmpty(Queue \*q) {

return q->front == q->rear;

}

// Hàm kiểm tra hàng đợi đầy

int isFull(Queue \*q) {

return q->rear == MAX\_SIZE;

}

// Hàm thêm phần tử vào hàng đợi

void enqueue(Queue \*q, int x) {

if (!isFull(q)) {

q->data[q->rear] = x;

q->rear++;

}

}

// Hàm xóa phần tử khỏi hàng đợi

int dequeue(Queue \*q) {

if (!isEmpty(q)) {

return q->data[q->front++];

}

return -1; // Trả về -1 nếu hàng đợi rỗng

}

// Hàm lấy số lượng phần tử trong hàng đợi

int size(Queue \*q) {

return q->rear - q->front;

}

int main() {

Queue q;

initializeQueue(&q);

int N;

scanf("%d", &N);

for (int i = 0; i < N; i++) {

char command[2]; // Để đọc thao tác 'E' hoặc 'D'

scanf("%s", command);

if (command[0] == 'E') {

int x;

scanf("%d", &x);

enqueue(&q, x);

printf("%d\n", size(&q)); // In ra số lượng phần tử sau khi thêm

} else if (command[0] == 'D') {

int removed = dequeue(&q);

if (removed == -1) {

printf("-1 %d\n", size(&q)); // In ra -1 và số lượng sau khi xóa

} else {

printf("%d %d\n", removed, size(&q)); // In ra phần tử đã xóa và số lượng sau khi xóa

}

}

}

return 0;

}

ỨNG DỤNG HÀNG ĐỢI

**Bằng cách sử dụng kiểu dữ liệu trừu tượng Stack và Queue đã cho, hãy viết chương trình kiểm tra một chuỗi ký tự có đọc xuôi và đọc ngược như nhau hay không (kiểm tra chuỗi palindrome).**

#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

#include <string.h>

#define MAX\_SIZE 1000 // Kích thước tối đa cho ngăn xếp và hàng đợi

// Định nghĩa kiểu dữ liệu ký tự

typedef char ElementType;

// Cấu trúc ngăn xếp

typedef struct {

ElementType data[MAX\_SIZE];

int top;

} Stack;

// Cấu trúc hàng đợi

typedef struct {

ElementType data[MAX\_SIZE];

int front, rear;

} Queue;

// Hàm khởi tạo ngăn xếp

void makenullStack(Stack \*pS) {

pS->top = -1;

}

// Kiểm tra ngăn xếp rỗng

int emptyStack(Stack S) {

return S.top == -1;

}

// Thêm phần tử vào ngăn xếp

void push(ElementType x, Stack \*pS) {

if (pS->top < MAX\_SIZE - 1) {

pS->data[++pS->top] = x;

}

}

// Lấy phần tử trên đỉnh ngăn xếp

ElementType top(Stack S) {

if (!emptyStack(S)) {

return S.data[S.top];

}

return '\0'; // Trả về giá trị mặc định nếu ngăn xếp rỗng

}

// Xóa phần tử trên đỉnh ngăn xếp

void pop(Stack \*pS) {

if (!emptyStack(\*pS)) {

pS->top--;

}

}

// Hàm khởi tạo hàng đợi

void makenullQueue(Queue \*pQ) {

pQ->front = 0;

pQ->rear = 0;

}

// Kiểm tra hàng đợi rỗng

int emptyQueue(Queue Q) {

return Q.front == Q.rear;

}

// Thêm phần tử vào hàng đợi

void enQueue(ElementType x, Queue \*pQ) {

if (pQ->rear < MAX\_SIZE) {

pQ->data[pQ->rear++] = x;

}

}

// Lấy phần tử ở đầu hàng đợi

ElementType front(Queue Q) {

if (!emptyQueue(Q)) {

return Q.data[Q.front];

}

return '\0'; // Trả về giá trị mặc định nếu hàng đợi rỗng

}

// Xóa phần tử ở đầu hàng đợi

void deQueue(Queue \*pQ) {

if (!emptyQueue(\*pQ)) {

pQ->front++;

}

}

// Hàm kiểm tra chuỗi palindrome

void checkPalindrome(const char \*input) {

Stack s;

Queue q;

makenullStack(&s);

makenullQueue(&q);

// Hiển thị chuỗi vừa nhập

printf("\"%s\"", input);

// Đọc từng ký tự trong chuỗi

for (int i = 0; input[i] != '\0'; i++) {

char ch = input[i];

// Chỉ lấy chữ cái

if (isalpha(ch)) {

char lowerCh = tolower(ch); // Chuyển về chữ thường

push(lowerCh, &s); // Đưa vào ngăn xếp

enQueue(lowerCh, &q); // Đưa vào hàng đợi

}

}

// So sánh các phần tử trong ngăn xếp và hàng đợi

while (!emptyQueue(q)) {

if (top(s) != front(q)) {

printf(" doc xuoi khac doc nguoc\n");

return;

}

pop(&s);

deQueue(&q);

}

printf(" doc xuoi va doc nguoc nhu nhau\n");

}

int main() {

char input[MAX\_SIZE];

// printf("Nhập chuỗi: ");

fgets(input, MAX\_SIZE, stdin); // Đọc chuỗi từ người dùng

input[strcspn(input, "\n")] = 0; // Loại bỏ ký tự newline

checkPalindrome(input); // Kiểm tra chuỗi

return 0;

}