void swap(int &x, int &y)

{

    int temp = x;

    x = y;

    y = temp;

}

void printArray(int arr[], int size)

{

    int i;

    for (i=0; i < size; i++)

        printf("%d ", arr[i]);

    printf("\n");

}

// Hàm sắp xếp bubble sort

void bubbleSort(int arr[], int n)

{

    int i, j;

    bool haveSwap = false;

    for (i = 0; i < n-1; i++){

    // i phần tử cuối cùng đã được sắp xếp

        haveSwap = false;

        for (j = 0; j < n-i-1; j++){

            if (arr[j] > arr[j+1]){

                swap(arr[j], arr[j+1]);

                haveSwap = true; // Kiểm tra lần lặp này có swap không

            }

        }

        // Nếu không có swap nào được thực hiện => mảng đã sắp xếp. Không cần lặp thêm

        if(haveSwap == false){

            break;

        }

    }}

void insertionSort(int arr[], int n)

{

   int i, key, j;

   for (i = 1; i < n; i++)

   {

       key = arr[i];

       j = i-1;

       /\* Di chuyển các phần tử có giá trị lớn hơn giá trị

       key về sau một vị trí so với vị trí ban đầu

       của nó \*/

       while (j >= 0 && arr[j] > key)

       {

           arr[j+1] = arr[j];

           j = j-1;

       }

       arr[j+1] = key;

   }

}

void selectionSort(int arr[], int n)

{

    int i, j, min\_idx;

    // Di chuyển ranh giới của mảng đã sắp xếp và chưa sắp xếp

    for (i = 0; i < n-1; i++)

    {

    // Tìm phần tử nhỏ nhất trong mảng chưa sắp xếp

    min\_idx = i;

    for (j = i+1; j < n; j++)

        if (arr[j] < arr[min\_idx])

        min\_idx = j;

    // Đổi chỗ phần tử nhỏ nhất với phần tử đầu tiên

        swap(arr[min\_idx], arr[i]);

    }

}

int binarySearch(int arr[], int l, int r, int x) {

if (r >= l) {

int mid = l + (r - l) / 2; // Tương đương (l+r)/2 nhưng ưu điểm tránh tràn số khi l,r lớn

// Nếu arr[mid] = x, trả về chỉ số và kết thúc.

if (arr[mid] == x)

return mid;

// Nếu arr[mid] > x, thực hiện tìm kiếm nửa trái của mảng

if (arr[mid] > x)

return binarySearch(arr, l, mid - 1, x);

// Nếu arr[mid] < x, thực hiện tìm kiếm nửa phải của mảng

return binarySearch(arr, mid + 1, r, x);

}

// Nếu không tìm thấy

return -1;

}

int main(void) {

int arr[] = { 2, 3, 4, 10, 40 };

int n = sizeof(arr) / sizeof(arr[0]);

int x = 10;

int result = binarySearch(arr, 0, n - 1, x);

if (result == -1)

printf("%d xuat hien tai chi so %d", x, result);

else

printf("%d xuat hien tai chi so %d", x, result);

return 0;

}

//Assignment 5: setup Linked List node

struct node

{

int data;

//int key;

struct node\* next;

struct node\* temp;

};

struct node\* head = NULL;

struct node\* current = NULL;

void push(struct node\*\* head\_ref, int data)

{

struct node\* new\_node = (struct node\*)malloc(sizeof(struct node));

new\_node->data = data;

new\_node->next = (\*head\_ref);

(\*head\_ref) = new\_node;

}

void printLinkedlist(struct node\* current)

{

//cach 1

/\*struct node\* ptr=head;

while (ptr != NULL)

{

printf("%d ", ptr->data);

ptr = ptr->next;

}\*/

////cach 1 cua thay Thao

//int index = 0;

//struct node\* current = head;

//while (current != NULL)

//{

// printf("node %d: %d\n", index++, current->data);

// current = current->next;

//}

//cach 2 cua thay Thao

//struct node\* current = head;

if (current == NULL)

return;

printf("Node data: %d\n", current->data);

printLinkedlist(current->next);

}

struct node\* read(int index)

{

/\*struct node\* current = head;

if (head = NULL) return NULL;

while(current->temp != index)

if()\*/

struct node\* cur = head;

for (int i = 0; i < index && cur != NULL; i++)

cur = cur->next;

/\*int i = 0;

while (i < index && cur != NULL)

{

cur = cur->next;

i++;

}\*/

return cur;

}

struct node\* search(int index)

{

struct node\* cur = head;

int i = 0;

/\*for (int i = 0; i < index && cur != NULL; i++)

cur = cur->data;\*/

return cur;

}

void insertTailR(struct node\* cur, int data)

{

//if (cur == NULL) {

// struct node\* node = (struct node\*)malloc(sizeof(struct node));

// node->data = data;

// node->next = NULL;

// head = node;

// return;

//}

if (cur->next == NULL) {

struct node\* node = (struct node\*)malloc(sizeof(struct node));

node->data = data;

node->next = NULL;

cur->next = node;

return;

}

insertTailR(cur->next, data);

}

void insertTail(struct node\* cur, int data)

{

struct node\* node = (struct node\*)malloc(sizeof(struct node));

node->data = data;

node->next = NULL;

//head = node;

//struct node\* cur = head;

while (cur->next != NULL)

cur = cur->next;

cur->next = node;

}

void insertHeadN(int data)

{

struct node\* node = (struct node\*)malloc(sizeof(struct node));

node->data = data;

node->next = head;

head = node;

}

void del(struct node\* before\_del)

{

struct node\* temp;

temp = before\_del->next;

before\_del->next = temp->next;

free(temp);

}

void deleteNode(struct node\*\* head\_ref, int key)

{

struct node\* temp = \*head\_ref;

struct node\* bef = (struct node\*)malloc(sizeof(struct node));

if (temp != NULL && temp->data == key)

{

\*head\_ref = temp->next;

free(temp);

return;

}

while (temp != NULL && temp->data != key)

{

bef = temp;

temp = temp->next;

}

if (temp == NULL) return;

bef->next = temp->next;

free(temp);

}

int main()

{

//struct node node0, node1,node2 ,node3;

//head = &node0;

//node0.data = 5;

//node0.next = NULL;

//node0.next = &node1;

//node1.data = 7;

//node1.next = NULL;

//node1.next = &node2;

//node2.data = 6;

//node2.next = NULL;

//node2.next = &node3;

////read

//printLinkedlist();

//struct node\* node = read(2);

//if (node != NULL)

// printf("%d \n", node->data);

//else printf("No date\n");

////printf("%p\n", head);

//malloc

/\*struct node\* node0 = NULL, \* node1 = NULL, \* node2 = NULL;

node0 = (struct node\*)malloc(sizeof(struct node));

node0->data = 5;

node0->next = NULL;

head = node0;

node1 = (struct node\*)malloc(sizeof(struct node));

node1->data = 7;

node1->next = NULL;

node0->next = node1;

node2 = (struct node\*)malloc(sizeof(struct node));

node2->data = 6;

node2->next = NULL;

node1->next = node2;\*/

push(&head, 5);

push(&head, 7);

push(&head, 6);

//printLinkedlist();

insertHeadN(8);

insertHeadN(1);

insertHeadN(3);

printLinkedlist(head);

printf("\n");

deleteNode(&head, 5);

insertTailR(head, 9);

printLinkedlist(head);

struct node\* node = read(1);

if (node != NULL)

printf("%d\n", node->data);

else printf("No data\n");

//node2->next = node3;

return 0;

}

//Assignment 4: stack and queue

#define MAX 100

int top;

int stack[MAX];

//int b = sizeof(stack) / sizeof(stack[0]);

//Struck

struct Stack {

int top;

int s[MAX];

};

void push\_s(struct Stack\* stack, int e)

{

if (stack->top < MAX - 1)

stack->top++;

stack->s[stack->top] = e;

}

int pop\_s(struct Stack\* stack)

{

if (stack->top != -1)

{

--stack->top;

return stack->s[stack->top + 1];

}

else printf("Stack is empty");

}

//int empty\_s(struct Stack\* stack)

//{

// return;

//}

void initStack(struct Stack\* stack);

struct Queue {

int head;

int tail;

int q[MAX];

};

void put(struct Queue\* queue, int e)

{

if (queue->tail == MAX)

printf("Full.\n");

else {

queue->q[queue->tail] = e;

queue->tail++;

}

/\*for (; queue->tail != MAX; queue->tail++)

queue->q[queue->tail] = e;\*/

}

int get(struct Queue\* queue)

{

if(queue->head < queue->tail)

{

//queue->q[queue->head] = 0;

queue->head++;

}

return queue->q[queue->head];

/\*int e;

if (queue->head < queue->tail) {

queue->head++;

e = queue->q[queue->head];

return e;

}\*/

}

bool q\_empty(struct Queue\* queue)

{

return (queue->head == queue->tail);

}

int Size\_q(struct Queue\* queue)

{

//return queue->q[queue->tail + 1];

return queue->tail - queue->head;

}

int Head(struct Queue\* queue)

{

return queue->q[queue->tail];

}

void checkqueue(struct Queue\* queue)

{

for (int i = 0; i < (queue->tail - queue->head); i++)

printf("%d ", queue->q[i]);

printf("\n");

}

//normal

//bool full()

//{

// if (top == MAX)

// return true;

// else return false;

//}

//bool empty()

//{

// if (top == -1)

// return true;

// else return false;

//}

//void push(int e)

//{

// if (!full())

// {

// top++;

// stack[top] = e;

// }

// else printf("Stack is full.\n");

//}

//int pop()

//{

// if (!empty())

// {

// //e = stack[top];

// --top;

// return stack[top + 1];

// }

// else printf("Stack is empty");

//}

//int Size()

//{

// return top++;

//}

int main()

{

/\*top = -1;

push(5);

push(6);

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

printf("%d\n", pop());\*/

struct Stack stack;

stack.top = -1;

push\_s(&stack, 5);

push\_s(&stack, 6);

printf("%d\n", pop\_s(&stack));

printf("%d\n", pop\_s(&stack));

struct Queue queue;

queue.head = -1;

queue.tail = 0;

put(&queue, 5);

put(&queue, 6);

printf("%d\n", get(&queue));

printf("%d\n", get(&queue));

}