Google Developer Student Clubs

Data structure & algorithm final exam preparation materials ✔️

Author: Ngo Van Trung - trung2601.it@gmail.com 👀

Sort algorithms 💻

1. **Bubble sort**

```
#include <stdio.h>

// Function swap two number
void swap(int *a, int *b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}

//Function bubble sort
void bubbleSort(int arr[], int n)
{
    for (int end = n - 1; end > 0; end--)
    {
        for (int i = 0; i < end; i++)
        {
            if (arr[i] > arr[i + 1])
            {
                swap(&arr[i], &arr[i + 1]);
            }
        }
    }
}

//Function print array
void printArray(int arr[], int n)
{
    for (int i = 0; i < n; i++)
    {
        printf("%d ", arr[i]);
    }
    printf("\n");
}

int main(int argc, char **argv)
{
    int arr[6] = {1, 6, 3, 23, 4, 19};
    int n = 6;
    // Print array before sort
    printArray(arr, n);
    // Run bubble sort
    bubbleSort(arr, n);
    // Print array after sort
    printArray(arr, n);
    return 0;
}
```

2. Selection Sort

```
#include <stdio.h>

void displayArray(int arr[], int n){
    for (int i = 0; i < n; i++){
        printf("%d ", arr[i]);
    }
    printf("\n");
}

void swap(int *a, int *b){
    int temp = *a;
    *a = *b;
    *b = temp;
}

void selectionSort(int arr[], int n){
    int position, temp;
    for (int i = 0 ; i < n - 1; i++){
        position = i;
        temp = arr[i];
        for (int j = i + 1; j < n; j++){
            if (arr[j] < temp){
                temp = arr[j];
                position = j;
            }
        }
        swap(&arr[i], &arr[position]);
    }
}

int main(){
    int arr[6] = {1, 6, 3, 23, 4, 19};
    int n = 6;
    // Print array before sort
    displayArray(arr, n);
    // Run selection sort
    selectionSort(arr,n);
    // Print array after sort
    displayArray(arr, n);
    return 0;
}
```

3. Insertion Sort

```
#include <stdio.h>

void displayArray(int arr[], int n){
    for (int i = 0; i < n; i++){
        printf("%d ", arr[i]);
    }
    printf("\n");
}

void insertionSort(int arr[], int n){
    int temp, position;
    for (int i = 1; i < n; i++){
        temp = arr[i];
        position = i;
        while (position >= 0 && arr[position - 1] > temp){
            arr[position] = arr[position - 1];
            position--;
        }
        arr[position] = temp;
    }
}

int main(){
    int arr[6] = {1, 6, 3, 23, 4, 19};
    int n = 6;
    // Print array before sort
    displayArray(arr, n);
    // Run insertion sort
    insertionSort(arr,n);
    // Print array after sort
    displayArray(arr, n);
    return 0;
}
```

4. Quick sort

a. Partition

```
// Function swap
void swap(int *a, int *b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}
// Function partition
int partition(int arr[], int left, int right)
{
    int pivotPosition = right;
    int pivot = arr[pivotPosition];
    right--;
    while (true)
    {
        while (left <= right && arr[left] < pivot)
            left++;
        while (right >= left && arr[right] > pivot)
            right--;
        if (left >= right)
            break;
        swap(&arr[left], &arr[right]);
        left++;
        right--;
    }
    swap(&arr[left], &arr[pivotPosition]);
    return left;
}
```

b. Quick sort

```
#include <stdio.h>
#include <stdbool.h>

//Function swap two number
void swap(int *a, int *b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}

//Function print array
void displayArray(int arr[], int *n)
{
    for (int i = 0; i < *n; i++)
    {
        printf("%d ", arr[i]);
    }
    printf("\n");
}

//Function partition
int partition(int arr[], int left, int right)
{
    int pivotPosition = right;
    int pivot = arr[pivotPosition];
    right--;
    while (true)
    {
        while (left <= right && arr[left] < pivot)
            left++;
        while (right >= left && arr[right] > pivot)
            right--;
        if (left >= right)
            break;
        swap(&arr[left], &arr[right]);
        left++;
        right--;
    }
    swap(&arr[left], &arr[pivotPosition]);
    return left;
}

//Function quick sort
void quickSort(int arr[], int left, int right)
{
    if (left >= right)
        return;
    int pivotPosition = partition(arr, left, right);
    quickSort(arr, left, pivotPosition - 1);
    quickSort(arr, pivotPosition + 1, right);
}

int main()
{
    int arr[] = {34, 10, 12, 30, 19, 42, 9, 37, 56, 27, 32};
    int length = sizeof(arr) / sizeof(arr[0]);
    // Print array before sort
    displayArray(arr, &length);
    // Run quick sort
    quickSort(arr,0,length - 1);
    // Print array after sort
    displayArray(arr, &length);
    return 0;
}
```

Binary search

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
```

```
int middle, step = 0;


int binarySearch(int arr[], int target, int left, int right){
    while (left <= right)
    {
        middle = (right + left) / 2;
        if (arr[middle] == target) {
            return middle;
        }
        else if (arr[middle] < target) {
            left = middle + 1;
        }
        else {
            right = middle - 1;
        }
    }
    return -1;

}

int main(){
    int arr[6] = {2, 3, 5, 6, 7};
    int n = 5;
    int left = 0, right = n - 1;
    int target = 2;
    int res = binarySearch(arr, target, left, right);
    if (res == -1){
        printf("Not found \n");
    }
    else {
        printf("Index of target is %d \n", res);
    }
    return 0;
}
```

Recursion

```
#include <stdio.h>
#include <stdlib.h>

void countDown(int n){
    if (n == 0) return;
    else{
        printf("%d\n", n);
        countDown(n - 1);
    }
}

int factorial(int n){
    if (n == 1) return 1;
    else return n * factorial(n - 1);
}

int main(){
    // countDown(10);
    printf("%d", factorial(5));
    return 0;
}
```

Stack & Queue

Stack

```
#include <stdio.h>

#define MAX 10

struct Stack {
    int data[MAX];
    int top;
};

void init(struct Stack *stack);
void push(struct Stack *stack, int value);
int pop(struct Stack *stack);
int top(struct Stack *stack);
int full(struct Stack *stack);
int empty(struct Stack *stack);
int size(struct Stack *stack);

void init(struct Stack *stack){
    stack->top = 0;
}

void push(struct Stack *stack, int value){
    stack->data[stack->top] = value;
    stack->top++;
}

int pop(struct Stack *stack){
    int res = stack->data[stack->top - 1];
    stack->top--;
    return res;
}

int top(struct Stack *stack){
    return stack->data[stack->top - 1];
}

int full(struct Stack *stack){ // return 1 : true, 0 : false
    return 1 ? stack->top == MAX : 0;
}
```

```
int empty(struct Stack *stack){ // return 1: true, 0 : false
    return 1 ? stack->top == 0 : 0;
}

int size(struct Stack *stack){
    return stack->top;
}

int main(){
    struct Stack stack;
    init(&stack);
    printf("Run Push \n");
    for (int i = 0; i <= MAX + 1; i++){
        if (!full(&stack)){
            push(&stack, i);
            printf("Size: %d ", size(&stack));
            printf("Push: %d \n", top(&stack));
        }
        else {
            printf("Stack is full \n");
        }
    }
    printf("Run Pop \n");
    for (int i = 0; i <= MAX; i++){
        if (!empty(&stack)){
            printf("Size: %d ", size(&stack));
            printf("Pop: %d \n", pop(&stack));
        }
        else {
            printf("Stack is empty \n");
        }
    }
    return 0;
}
```

Queue

```
#include <stdio.h>
#define MAX 10

struct Queue{
    int data[MAX];
    int head;
    int tail;
};

void init(struct Queue* queue){
    queue->head = queue->tail = 0;
}

void put(struct Queue* queue, int value){
    queue->data[queue->tail] = value;
    queue->tail++;
}
```

```
int get(struct Queue* queue){
    int res = queue->data[queue->head];
    queue->head++;
    return res;
}

int isEmpty(struct Queue* queue){ // 1: true , 0 : false
    return 1 ? queue->head == queue->tail : 0;
}

int isFull(struct Queue* queue){ // 1: true , 0 : false
    return 1 ? queue->tail == MAX : 0;
}

int size(struct Queue* queue){
    return queue->tail - queue->head;
}

int main(){
    struct Queue queue;
    init(&queue);
    printf("Run put \n");
    for (int i = 0; i <= MAX + 1; i++){
        if (!isFull(&queue)) {
            put(&queue, i);
            printf("Size: %d ", size(&queue));
            printf("Put: %d \n", i);
        }
        else {
            printf("Queue is full \n");
        }
    }
    printf("Run get \n");
    for (int i = 0; i <= MAX; i++){
        if (!isEmpty(&queue)){
            printf("Size: %d ", size(&queue));
            printf("Get: %d \n", get(&queue));
        }
        else {
            printf("Queue is empty \n");
        }
    }
    return 0;
}
```

Linked list

```
/**
 * @file LinkedList.c
 * @author Ngo Van Trung (trung2601.it@gmail.com)
 * @brief Implement data structure Linked list
 * @version 0.1
 * @date 2022-12-02
 *
 * @copyright Copyright (c) 2022
 *
 */
#include <stdio.h>
#include <stdlib.h>

struct Node
{
    int data;          // Data of curren node
    struct Node *next; // Contain address of next node
};

// Function create new node
struct Node *makeNode(int value)
{
    struct Node *newNode;
    newNode = (struct Node *)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->next = NULL;
    return newNode;
}

// Function print linked list
void displayList(struct Node *head)
{
    while (head != NULL)
    {
        printf("%d -> ", head->data);
        head = head->next;
    }
    printf("NULL\n");
}

// Function insert new node to linked list
void insert(struct Node **head, int value, int position)
{
    // Case insert in head of linked list
    if (position - 1 == 0)
    {
        struct Node *newNode = makeNode(value);
        newNode->next = *head;
        *head = newNode;
        return;
    }

    // Other case
    int count = 0;
    struct Node *temp = *head;
    while (count < position - 1)
    {
        temp = temp->next;
        count++;
    }
    struct Node *newNode = makeNode(value);
    newNode->next = temp->next;
    temp->next = newNode;
    return;
}

void insertTail(struct Node **head, int data)
{
    if (*head == NULL || (*head)->next == NULL)
    {
        struct Node *newNode = makeNode(data);
        if (*head == NULL)
            *head = newNode;
        else
        {
            (*head)->next = newNode;
        }
    }
    else
    {
        insertTail(&((*head)->next), data);
    }
}

// Function search element by data in linked list
int search(struct Node *head, int value)
{
    int res = 0;
    while (head != NULL)
    {
        if (head->data == value)
            return res;
        res++;
        head = head->next;
    }
    return -1;
}

// Function to delete data by position of linked list
void delete(struct Node *head, int position)
{
    int index = 0;
    while (index != position - 1)
    {
        head = head->next;
        index++;
    }
    head->next = head->next->next;
    return;
}

int main()
{
    struct Node *head, *temp = head;
    head = NULL; // Define node head is NULL
    // Input data to linked list
    for (int i = 0; i < 10; i++)
    {
        temp = makeNode(i);
        temp->next = head;
        head = temp;
```

```c
    }
    // Demo methods
    displayList(head);
    insert(&head, 13, 3);
    displayList(head);
    printf("Search: %d\n", search(head, 4));
    delete (head, 5);
    displayList(head);
    insertTail(&head, 100);
    displayList(head);
    return 0;
}
```

Queue by linked list

```c
/**
 * @file QueueByLinkedList.c
 * @author Ngo Van Trung (trung2601.it@gmail.com)
 * @brief
 * @version 0.1
 * @date 2022-12-05
 *
 * @copyright Copyright (c) 2022
 *
 */

#include <stdio.h>
#include <stdlib.h>

typedef struct Node
{
    int data;
    struct Node *next;
} Node;

typedef struct Queue
{
    Node *head;
    Node *tail;
} Queue;

Node *makeNode(int value)
{
    Node *newNode = (Node *)malloc(sizeof(Node));
    newNode->data = value;
    newNode->next = NULL;
    return newNode;
}

Queue *createQueue()
{
    Queue *newQueue = (Queue *)malloc(sizeof(Queue));
    newQueue->head = newQueue->tail = NULL;
    return newQueue;
}

void put(Queue *q, int value)
{
    Node *temp = makeNode(value);
    if (q->head == NULL)
    {
        q->head = temp;
        q->tail = temp;
    }
    else
    {
        q->tail->next = temp;
        q->tail = temp;
    }
    return;
}

int get(Queue *q)
{
    if (q->head == NULL)
    {
        printf("Queue is empty");
        return -1;
    }
    Node *temp = q->head;
    int data = temp->data;
    q->head = q->head->next;
    if (q->head == NULL)
    {
        q->tail = NULL;
    }
    free(temp);
    return data;
}

int main()
{
    Queue *q = createQueue();
    put(q, 19);
    put(q, 6);
    put(q, 23);
    printf("Get: %d\n", get(q));
    printf("%d\n", q->head->data);
    printf("%d\n", q->tail->data);
    return 0;
}
```

Binary tree

```
/**
 * @file BinaryTree.c
 * @author Ngo Van Trung (trung2601.it@gmail.com)
 * @brief
 * @version 0.1
 * @date 2022-12-02
 *
 * @copyright Copyright (c) 2022
 *
 */

#include <stdio.h>
#include <stdlib.h>

typedef struct Node
{
    int data;
    struct Node *left;
    struct Node *right;
} Node;

Node *makeNode(int value)
{
    Node *newNode;
    newNode = (Node *)malloc(sizeof(Node));
    newNode->data = value;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}

void display(Node *head)
{
    if (head != NULL)
    {
        display(head->left);
        printf("%d ", head->data);
        display(head->right);
    }
}

Node *searchTree(Node *root, int target)
{
    if (root == NULL)
        return NULL;
    if (root->data == target)
        return root;
    else if (root->data > target)
        return searchTree(root->left, target);
    else
        return searchTree(root->right, target);
}

int main()
{
    Node *head = makeNode(50);
    head->left = makeNode(25);
    head->right = makeNode(75);
    head->left->right = makeNode(33);
    // display(head);
    Node *search = searchTree(head, 33);
    int res = search != NULL ? search->data : -1;
    printf("%d\n", res);
    free(head);
    return 0;
}
```