Лабораторна робота 8(4)

Варіант 18

1. 1.4

#include <iostream>

template<typename T>

double average(T\* array, int size) {

double sum = 0;

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

sum += array[i];

}

return sum / size;

}

template<>

double average<char\*>(char\*\* array, int size) {

double sum = 0;

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

}

return sum / size;

}

int main() {

int intArray[] = { 1, 2, 3, 4, 5 };

std::cout << "Average of intArray: " << average(intArray, 5) << std::endl;

double doubleArray[] = { 1.1, 2.2, 3.3, 4.4, 5.5 };

std::cout << "Average of doubleArray: " << average(doubleArray, 5) << std::endl;

return 0;

}

1. 2.4

#include <iostream>

#include <cstring>

template<typename T>

void selectionSort(T\* array, int size) {

for (int i = 0; i < size - 1; i++) {

int minIndex = i;

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

if (array[j] < array[minIndex]) {

minIndex = j;

}

}

if (minIndex != i) {

std::swap(array[i], array[minIndex]);

}

}

}

template<>

void selectionSort<char\*>(char\*\* array, int size) {

for (int i = 0; i < size - 1; i++) {

int minIndex = i;

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

if (strcmp(array[j], array[minIndex]) < 0) {

minIndex = j;

}

}

if (minIndex != i) {

std::swap(array[i], array[minIndex]);

}

}

}

int main() {

int intArray[] = { 3, 1, 4, 1, 5, 9, 2, 6, 5, 3 };

int intSize = sizeof(intArray) / sizeof(int);

std::cout << "Before sorting: ";

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

std::cout << intArray[i] << " ";

}

std::cout << std::endl;

selectionSort(intArray, intSize);

std::cout << "After sorting: ";

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

std::cout << intArray[i] << " ";

}

std::cout << std::endl;

char\* stringArray[] = { "hello", "world", "apple", "banana" };

int stringSize = sizeof(stringArray) / sizeof(char\*);

std::cout << "Before sorting: ";

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

std::cout << stringArray[i] << " ";

}

std::cout << std::endl;

selectionSort(stringArray, stringSize);

std::cout << "After sorting: ";

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

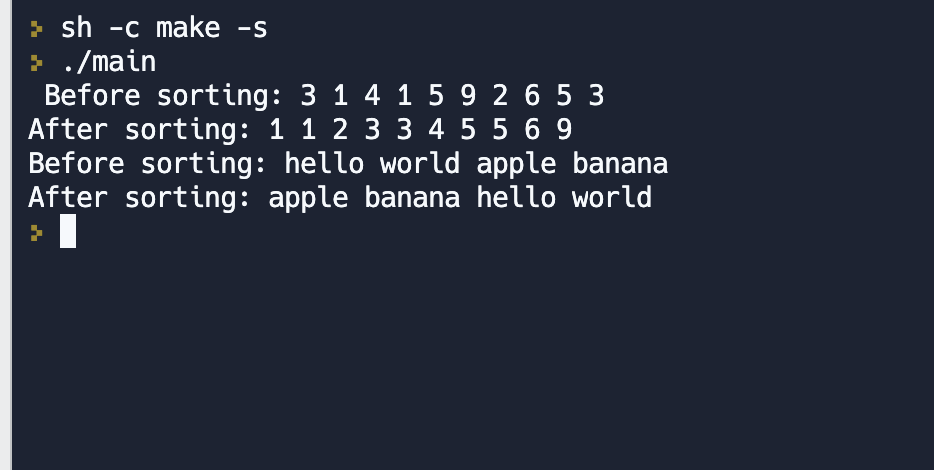
std::cout << stringArray[i] << " ";

}

std::cout << std::endl;

return 0;

}



1. 3.4

#include <iostream>

using namespace std;

const int MAXSIZE = 100;

template<typename T>

class CircularQueue {

private:

int front, rear, count;

T data[MAXSIZE];

public:

CircularQueue() {

front = rear = count = 0;

}

void enqueue(T item) {

if (count == MAXSIZE) {

cout << "Queue is full!" << endl;

return;

}

data[rear] = item;

rear = (rear + 1) % MAXSIZE;

count++;

}

void dequeue() {

if (count == 0) {

cout << "Queue is empty!" << endl;

return;

}

front = (front + 1) % MAXSIZE;

count--;

}

T peek() {

if (count == 0) {

cout << "Queue is empty!" << endl;

return T();

}

return data[front];

}

bool isEmpty() {

return count == 0;

}

bool isFull() {

return count == MAXSIZE;

}

};

int main() {

CircularQueue<int> q;

q.enqueue(1);

q.enqueue(2);

q.enqueue(3);

q.enqueue(4);

q.enqueue(5);

while (!q.isEmpty()) {

cout << q.peek() << " ";

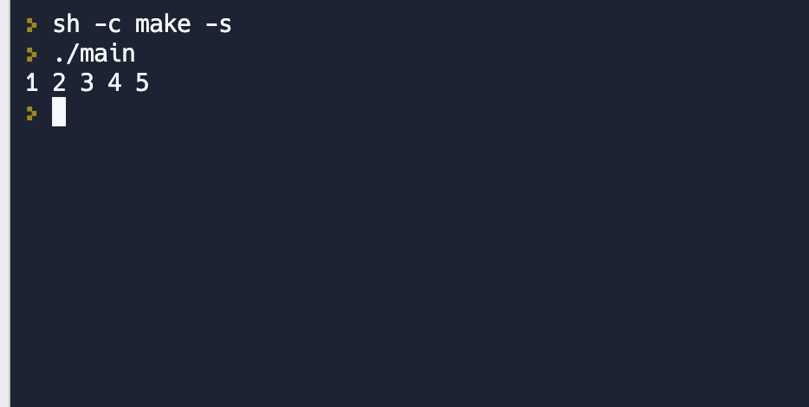
q.dequeue();

}

cout << endl;

return 0;

}



1. 4.3

#include <iostream>

template <typename T>

class DoublyLinkedList {

private:

struct Node {

T data;

Node\* next;

Node\* prev;

Node(const T& d, Node\* n = nullptr, Node\* p = nullptr) : data(d), next(n), prev(p) {}

};

Node\* head;

Node\* tail;

int size;

public:

DoublyLinkedList() : head(nullptr), tail(nullptr), size(0) {}

void push\_back(const T& data) {

Node\* node = new Node(data, nullptr, tail);

if (tail != nullptr) {

tail->next = node;

}

tail = node;

if (head == nullptr) {

head = node;

}

++size;

}

void push\_front(const T& data) {

Node\* node = new Node(data, head);

if (head != nullptr) {

head->prev = node;

}

head = node;

if (tail == nullptr) {

tail = node;

}

++size;

}

T& front() {

return head->data;

}

T& back() {

return tail->data;

}

int getSize() {

return size;

}

class Iterator {

private:

Node\* current;

public:

Iterator(Node\* node) : current(node) {}

T& operator\*() const {

return current->data;

}

Iterator& operator++() {

current = current->next;

return \*this;

}

Iterator& operator--() {

current = current->prev;

return \*this;

}

bool operator==(const Iterator& other) const {

return current == other.current;

}

bool operator!=(const Iterator& other) const {

return current != other.current;

}

};

Iterator begin() {

return Iterator(head);

}

Iterator end() {

return Iterator(nullptr);

}

};

int main() {

DoublyLinkedList<int> list;

list.push\_back(1);

list.push\_back(2);

list.push\_back(3);

list.push\_front(0);

for (DoublyLinkedList<int>::Iterator it = list.begin(); it != list.end(); ++it) {

std::cout << \*it << " ";

}

std::cout << std::endl;

return 0;

}

