Министерство науки и высшего образования

Пензенский государственный университет

Кафедра “Вычислительная техника”

**Отчет**

по лабораторной работе №3

по курсу “ Логика и основы алгоритмизации в инженерных задачах”

на тему “Динамические списки”

Выполнили

студенты группы 22ВВП2:

Тельнов И.В.

Городничев М.И.

Шнайдер К.С.

Приняли

Юрова О.В.

Акифьев И.В.

Пенза 2023

**Лабораторное задание**

1. Реализовать приоритетную очередь, путём добавления элемента в список в соответствии с приоритетом объекта (т.е. объект с большим приоритетом становится перед объектом с меньшим приоритетом).
2. На основе приведенного кода реализуйте структуру данных Очередь.
3. На основе приведенного кода реализуйте структуру данных Стек.

**Листинг**

**Задание 1**:

**#include <locale.h>**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <string.h>**

**char inputFileName[] = "queue.txt";**

**char outputFileName[] = "queue.txt";**

**struct node {**

**char inf[256]; // дданные**

**int priority; // приоритет**

**struct node\* next; // указатель на следующий элемент**

**};**

**struct node\* head = NULL, \*last = NULL, \*f = NULL; // указатели на первый и последний элементы**

**int dlinna = 0;**

**struct node\* get\_struct(void) {**

**struct node\* p = NULL;**

**char s[256];**

**int priority;**

**if ((p = (struct node\*)malloc(sizeof(struct node))) == NULL) { // выделяем память под новый элемент**

**printf("Memory allocation error\n");**

**exit(1);**

**}**

**printf("Enter object name: \n"); // вводим данные**

**scanf("%s", s);**

**if (\*s == 0) {**

**printf("No recording was made\n");**

**return NULL;**

**}**

**strcpy(p->inf, s);**

**printf("Enter object name: \n"); // вводим приоритет**

**scanf("%d", &priority);**

**p->priority = priority;**

**p->next = NULL;**

**return p; // возвращаем указатель на созданный элемент**

**}**

**/\* Последовательное добавление в список элемента (в соответствии с приоритетом) \*/**

**void spstore(void) {**

**struct node\* p = NULL;**

**p = get\_struct();**

**if (head == NULL && p != NULL) { // если списка нет, то устанавливаем голову списка**

**head = p;**

**last = p;**

**}**

**else if (head != NULL && p != NULL) { // список уже есть, то вставляем в соответствии с приоритетом**

**struct node\* current = head;**

**struct node\* prev = NULL;**

**while (current != NULL && p->priority >= current->priority) {**

**prev = current;**

**current = current->next;**

**}**

**if (prev == NULL) {**

**// Вставляем в начало списка**

**p->next = head;**

**head = p;**

**}**

**else {**

**// Вставляем между prev и current**

**prev->next = p;**

**p->next = current;**

**if (current == NULL) {**

**last = p; // Если p вставлен в конец списка, обновляем last**

**}**

**}**

**}**

**}**

**/\* Просмотр содержимого списка. \*/**

**void review(void) {**

**struct node\* struc = head;**

**if (head == NULL) {**

**printf("List is empty\n");**

**}**

**while (struc) {**

**printf("Name - %s, Priority - %d\n", struc->inf, struc->priority);**

**struc = struc->next;**

**}**

**}**

**/\* Поиск элемента по содержимому. \*/**

**struct node\* find(char\* name) {**

**struct node\* struc = head;**

**if (head == NULL) {**

**printf("List is empty\n");**

**}**

**while (struc) {**

**if (strcmp(name, struc->inf) == 0) {**

**return struc;**

**}**

**struc = struc->next;**

**}**

**printf("Element not found\n");**

**return NULL;**

**}**

**/\* Удаление элемента по содержимому. \*/**

**void del(char\* name) {**

**struct node\* struc = head; // указатель, проходящий по списку установлен на начало списка**

**struct node\* prev = NULL; // указатель на предшествующий удаляемому элементу**

**int flag = 0; // индикатор отсутствия удаляемого элемента в списке**

**if (head == NULL) // если голова списка равна NULL, то список пуст**

**{**

**printf("List is empty\n");**

**return;**

**}**

**if (strcmp(name, struc->inf) == 0) // если удаляемый элемент - первый**

**{**

**flag = 1;**

**head = struc->next; // устанавливаем голову на следующий элемент**

**free(struc); // удаляем первый элемент**

**struc = head; // устанавливаем указатель для продолжения поиска**

**}**

**else {**

**prev = struc;**

**struc = struc->next;**

**}**

**while (struc) // проход по списку и поиск удаляемого элемента**

**{**

**if (strcmp(name, struc->inf) == 0) // если нашли, то**

**{**

**flag = 1; // выставляем индикатор**

**if (struc->next) // если найденный элемент не последний в списке**

**{**

**prev->next = struc->next; // меняем указатели**

**free(struc); // удаляем элемент**

**struc = prev->next; // устанавливаем указатель для продолжения поиска**

**}**

**else // если найденный элемент последний в списке**

**{**

**prev->next = NULL; // обнуляем указатель предшествующего элемента**

**free(struc); // удаляем элемент**

**last = prev; // обновляем last**

**return;**

**}**

**}**

**else // если не нашли, то**

**{**

**prev = struc; // устанавливаем указатели для продолжения поиска**

**struc = struc->next;**

**}**

**}**

**if (flag == 0) // если флаг = 0, значит нужный элемент не найден**

**{**

**printf("Element not found\n");**

**return;**

**}**

**}**

**void saveQueueToFile(const char\* filename) {**

**FILE\* file = fopen(filename, "w");**

**if (file == NULL) {**

**printf("Open file error\n");**

**return;**

**}**

**struct node\* struc = head;**

**while (struc) {**

**fprintf(file, "%s %d\n", struc->inf, struc->priority);**

**struc = struc->next;**

**}**

**fclose(file);**

**printf("Queue was successfully saved %s\n", filename);**

**}**

**void loadQueueFromFile(const char\* filename) {**

**FILE\* file = fopen(filename, "r");**

**if (file == NULL) {**

**printf("Open file error\n");**

**return;**

**}**

**// Очистка существующей очереди**

**struct node\* current = head;**

**while (current != NULL) {**

**struct node\* next = current->next;**

**free(current);**

**current = next;**

**}**

**head = last = NULL;**

**char inf[256];**

**int priority;**

**while (fscanf(file, "%s %d", inf, &priority) != EOF) {**

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

**if (p == NULL) {**

**printf("Memory allocation error\n");**

**fclose(file);**

**exit(1);**

**}**

**strcpy(p->inf, inf);**

**p->priority = priority;**

**p->next = NULL;**

**// Вставка элемента в отсортированную очередь**

**if (head == NULL || priority < head->priority) {**

**// Вставка в начало**

**p->next = head;**

**head = p;**

**}**

**else {**

**struct node\* current = head;**

**struct node\* prev = NULL;**

**while (current != NULL && priority >= current->priority) {**

**prev = current;**

**current = current->next;**

**}**

**// Вставка между prev и current**

**prev->next = p;**

**p->next = current;**

**if (current == NULL) {**

**last = p; // Обновление last, если p вставлен в конец**

**}**

**}**

**}**

**fclose(file);**

**printf("Queue was successfully upload %s\n", filename);**

**}**

**int main() {**

**int choice;**

**char name[256];**

**struct node\* found = NULL;**

**while (1) {**

**printf("\n1. Add element into a queue\n");**

**printf("2. View queue\n");**

**printf("3. Find element\n");**

**printf("4. Delete element\n");**

**printf("5. Save queue to file\n");**

**printf("6. Load a queue from a file\n");**

**printf("7. Exit\n");**

**printf("Enter operation: ");**

**scanf("%d", &choice);**

**switch (choice) {**

**case 1:**

**spstore();**

**break;**

**case 2:**

**review();**

**break;**

**case 3:**

**printf("Enter the name of the object to search for: ");**

**scanf("%s", name);**

**found = find(name);**

**if (found != NULL) {**

**printf("Element: Name - %s, Priority - %d\n", found->inf, found->priority);**

**}**

**break;**

**case 4:**

**printf("Enter the name of the object to delete: ");**

**scanf("%s", name);**

**del(name);**

**break;**

**case 5:**

**saveQueueToFile(outputFileName); // Function to save the queue to a file**

**break;**

**case 6:**

**loadQueueFromFile(inputFileName); // Function to load the queue from a file**

**break;**

**case 7:**

**exit(0);**

**default:**

**printf("Wrong choice. Please select again.\n");**

**}**

**}**

**}**

**Задание 2**:

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <string.h>**

**#include <locale.h>**

**struct node**

**{**

**char inf[256]; // Данные**

**struct node \*next; // Ссылка на следующий элемент**

**};**

**struct node \*front = NULL, \*rear = NULL; // Указатели на начало и конец очереди**

**void enqueue(void); // Добавление элемента в очередь**

**void dequeue(void); // Удаление элемента из очереди**

**void display(void); // Просмотр содержимого очереди**

**void saveToFile(void); // Сохранение данных очереди в файл**

**void loadFromFile(void); // Загрузка данных очереди из файла**

**struct node \*get\_struct(void)**

**{**

**struct node \*p = NULL;**

**char s[256];**

**if ((p = (struct node \*)malloc(sizeof(struct node))) == NULL) // Выделяем память под новый элемент очереди**

**{**

**printf("Memory allocation mistake\n");**

**exit(1);**

**}**

**printf("Enter object name: \n"); // Вводим данные**

**scanf("%s", s);**

**if (\*s == 0)**

**{**

**printf("No recording was made\n");**

**return NULL;**

**}**

**strcpy(p->inf, s);**

**p->next = NULL;**

**return p; // Возвращаем указатель на созданный элемент**

**}**

**/\* Добавление элемента в очередь \*/**

**void enqueue(void)**

**{**

**struct node \*p = NULL;**

**p = get\_struct();**

**if (rear == NULL && p != NULL) // Если очередь пуста, то устанавливаем начало и конец очереди**

**{**

**front = p;**

**rear = p;**

**}**

**else if (rear != NULL && p != NULL) // В очереди уже есть элементы, добавляем в конец**

**{**

**rear->next = p;**

**rear = p;**

**}**

**return;**

**}**

**/\* Удаление элемента из очереди (из начала) \*/**

**void dequeue(void)**

**{**

**if (front == NULL) // Если начало очереди равно NULL, то очередь пуста**

**{**

**printf("Queue is empty\n");**

**return;**

**}**

**struct node \*temp = front;**

**front = front->next;**

**free(temp);**

**if (front == NULL) // Если после удаления элемента начало очереди стало NULL, то это был последний элемент**

**{**

**rear = NULL; // Устанавливаем и конец очереди в NULL**

**}**

**}**

**/\* Просмотр содержимого очереди \*/**

**void display(void)**

**{**

**struct node \*struc = front;**

**if (front == NULL)**

**{**

**printf("Queue is empty\n");**

**}**

**while (struc)**

**{**

**printf("%s \n", struc->inf);**

**struc = struc->next;**

**}**

**return;**

**}**

**/\* Сохранение данных очереди в файл \*/**

**void saveToFile(void)**

**{**

**FILE \*file;**

**struct node \*struc = front;**

**if (front == NULL)**

**{**

**printf("Queue is empty, no files to save\n");**

**return;**

**}**

**file = fopen("queue.txt", "w");**

**if (file == NULL)**

**{**

**printf("Open file error\n");**

**return;**

**}**

**while (struc)**

**{**

**fprintf(file, "%s\n", struc->inf);**

**struc = struc->next;**

**}**

**fclose(file);**

**printf("Queue data has been successfully saved in a file\n");**

**}**

**/\* Загрузка данных очереди из файла \*/**

**void loadFromFile(void)**

**{**

**FILE \*file;**

**char s[256];**

**file = fopen("queue.txt", "r");**

**if (file == NULL)**

**{**

**printf("File not found\n");**

**return;**

**}**

**while (fscanf(file, "%s", s) != EOF)**

**{**

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

**if (p == NULL)**

**{**

**printf("Memory allocation error\n");**

**exit(1);**

**}**

**strcpy(p->inf, s);**

**p->next = NULL;**

**if (rear == NULL)**

**{**

**front = p;**

**rear = p;**

**}**

**else**

**{**

**rear->next = p;**

**rear = p;**

**}**

**}**

**fclose(file);**

**printf("Data from the file has been successfully uploaded to the queue\n");**

**}**

**int main()**

**{**

**int choice;**

**while (1)**

**{**

**printf("\n Queue operations:\n");**

**printf("1. Add element\n");**

**printf("2. Delete element\n");**

**printf("3. View queue\n");**

**printf("4. Sava data to file\n");**

**printf("5. Load queue data from a file\n");**

**printf("6. Exit\n");**

**printf("Enter operation: ");**

**scanf("%d", &choice);**

**switch (choice)**

**{**

**case 1:**

**enqueue();**

**break;**

**case 2:**

**dequeue();**

**break;**

**case 3:**

**display();**

**break;**

**case 4:**

**saveToFile();**

**break;**

**case 5:**

**loadFromFile();**

**break;**

**case 6:**

**exit(0);**

**default:**

**printf("Wrong choice\n");**

**}**

**}**

**}**

**Задание 3**:

**#define \_CRT\_SECURE\_NO\_WARNINGS**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <string.h>**

**#include <locale.h>**

**struct node**

**{**

**char inf[256]; // Полезная информация**

**struct node\* next; // Ссылка на следующий элемент**

**};**

**struct node\* top = NULL; // Указатель на вершину стека**

**void push(void); // Добавление элемента в стек**

**void pop(void); // Удаление элемента из стека**

**void display(void); // Просмотр содержимого стека**

**void saveToFile(void); // Сохранение данных стека в файл**

**void loadFromFile(void); // Загрузка данных стека из файла**

**struct node\* get\_struct(void)**

**{**

**struct node\* p = NULL;**

**char s[256];**

**if ((p = (struct node\*)malloc(sizeof(struct node))) == NULL) // Выделяем память под новый элемент стека**

**{**

**printf("Memory allocation error\n");**

**exit(1);**

**}**

**printf("Enter object name: \n"); // Вводим данные**

**scanf("%s", s);**

**if (\*s == 0)**

**{**

**printf("The recording was not completed\n");**

**return NULL;**

**}**

**strcpy(p->inf, s);**

**p->next = NULL;**

**return p; // Возвращаем указатель на созданный элемент**

**}**

**/\* Добавление элемента в стек \*/**

**void push(void)**

**{**

**struct node\* p = NULL;**

**p = get\_struct();**

**if (p != NULL)**

**{**

**p->next = top;**

**top = p;**

**}**

**}**

**/\* Удаление элемента из стека (с вершины) \*/**

**void pop(void)**

**{**

**if (top == NULL)**

**{**

**printf("Stack is empty\n");**

**return;**

**}**

**struct node\* temp = top;**

**top = top->next;**

**free(temp);**

**}**

**/\* Просмотр содержимого стека \*/**

**void display(void)**

**{**

**struct node\* struc = top;**

**if (top == NULL)**

**{**

**printf("Stack is empty\n");**

**}**

**while (struc)**

**{**

**printf("%s \n", struc->inf);**

**struc = struc->next;**

**}**

**}**

**/\* Сохранение данных стека в файл \*/**

**void saveToFile(void)**

**{**

**FILE\* file;**

**struct node\* struc = top;**

**if (top == NULL)**

**{**

**printf("Stack is empty, no data to save\n");**

**return;**

**}**

**file = fopen("stack.txt", "w");**

**if (file == NULL)**

**{**

**printf("Open file error\n");**

**return;**

**}**

**while (struc)**

**{**

**fprintf(file, "%s\n", struc->inf);**

**struc = struc->next;**

**}**

**fclose(file);**

**printf("Stack data was successfully save\n");**

**}**

**void loadFromFile(void)**

**{**

**FILE\* file;**

**char s[256];**

**file = fopen("stack.txt", "r");**

**if (file == NULL)**

**{**

**printf("File not found\n");**

**return;**

**}**

**while (fscanf(file, "%s", s) != EOF)**

**{**

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

**if (p == NULL)**

**{**

**printf("File allocation error\n");**

**exit(1);**

**}**

**strcpy(p->inf, s);**

**p->next = top;**

**top = p;**

**}**

**fclose(file);**

**printf("Data was successfully loaded\n");**

**}**

**int main()**

**{**

**int choice;**

**while (1)**

**{**

**printf("\n Stack operation:\n");**

**printf("1. Add element\n");**

**printf("2. Delete element\n");**

**printf("3. View stack\n");**

**printf("5. Save data\n");**

**printf("6. Exit\n");**

**printf("Enter operation: ");**

**scanf("%d", &choice);**

**switch (choice)**

**{**

**case 1:**

**push();**

**break;**

**case 2:**

**pop();**

**break;**

**case 3:**

**display();**

**break;**

**case 4:**

**saveToFile();**

**break;**

**case 5:**

**loadFromFile();**

**break;**

**case 6:**

**exit(0);**

**default:**

**printf("Wrong choice\n");**

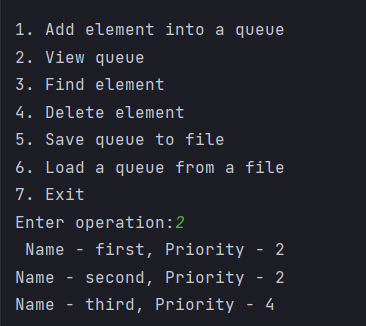
**}**

**}**

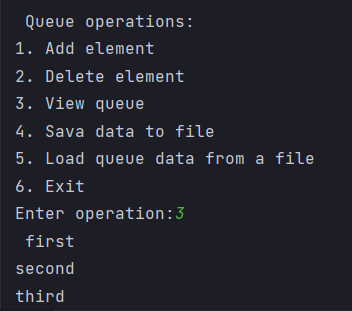
**}**

**Результаты работы программы**

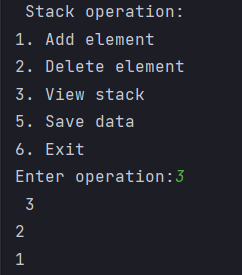
**Задание 1**:



**Задание 2**:



**Задание 3**:



**Вывод**

В ходе выполнения данной лабораторной работы были получены практические навыки реализации базовых динамических структур данных - очереди, стека и приоритетной очереди на основе односвязных списков на языке C. Были изучены основные принципы организации односвязных списков, реализованы функции для работы с динамической памятью при создании и удалении элементов.