Таганрог 2023

Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное

учреждение высшего образования

«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

(ФГАОУ ВО «ЮФУ»)

Институт компьютерных технологий и информационной безопасности

Кафедра системного анализа и телекоммуникаций

**ЛАБОРАТОРНАЯ РАБОТА № 2**

по дисциплине «Основы алгоритмизации и программирования»

Вариант 2

на тему:

«Работа со структурами»

Выполнила

студентка группы КТбо1-2 Н. А. Свидич

Принял

доцент кафедры САиТ В. С. Лапшин

**Содержание**

Содержание 2

Техническое задание 3

Цель работы 3

Задача 3

Ход работы 4

Алгоритмы 4

Блок-схемы 5

Написание программы 6

Примеры работы программы 7

Вывод 9

Листинг (Приложение а) 10

**Техническое задание**

**Цель работы**

Целью данной лабораторной работы является освоение студентами способов представления и обработки данных в виде списков.

Для успешного выполнения лабораторной работы необходимо владеть теоретическими сведениями по таким понятиям, как списки, указатели, способы выделения памяти, 16 функции, отвечающие за выделение памяти, функции mallocO и callocO и их отличия, функция free(), структуры данных, информационная и адресная составляющие списка.

**Задача**

Вариант 2.

Задание. Написать программы для работы с односвязным и двусвязным списками в соответствии с выданным вариантом задания. Предусмотреть в программах следующие функции:

1. Включить новый элемент в конец списка.

2. Включить новый элемент на заданное пользователем

место списка.

3. Включить новый элемент после элемента с заданной

информационной частью.

4. Включить новый элемент перед элементом с заданной

информационной частью.

5. Включить новый элемент в середину списка.

6. Исключить элемент из середины списка.

7. Исключить элемент с заданной информационной

частью.

8. Исключить элемент из конца списка.

9. Исключить элемент из заданного пользователем места

списка.

10. Исключить элемент из головы списка.

Программы должны уметь обрабатывать структуру данных в соответствии с вариантом заданий. Задание варианта: Карточка студента и его успеваемости деканата.

**Ход работы**

**Алгоритмы**

1. Определение структуры данных:

Создается структура Student, содержащая информацию о студенте: имя, идентификатор, средний балл (GPA) и указатель на следующего студента в списке.

1. Функции для работы со списком:

createStudent: создает новый элемент списка с заданными параметрами.

printStudents: выводит информацию о всех студентах в списке.

appendStudent: добавляет нового студента в конец списка.

insertAtPosition: вставляет нового студента на указанную позицию в списке.

insertAfter: вставляет нового студента после студента с заданным именем.

insertBefore: вставляет нового студента перед студентом с заданным именем.

insertInMiddle: вставляет нового студента в середину списка.

deleteFromEnd: удаляет студента с конца списка.

deleteAtPosition: удаляет студента с указанной позиции.

deleteFromHead: удаляет студента с начала списка.

deleteByInfo: удаляет студента по заданному имени.

deleteFromMiddle: удаляет студента из середины списка.

1. Меню для пользователя:

Пользователю предлагается меню с различными функциями для управления списком студентов.

Пользователь вводит номер функции, которую хочет выполнить.

1. Основной цикл программы:

В цикле while программа запрашивает у пользователя выбор функции и выполняет соответствующие действия.

Для некоторых функций требуется ввод дополнительной информации, такой как имя студента, идентификатор, GPA, позиция в списке или имя для поиска.

1. Завершение работы:

Если пользователь выбирает функцию "Выйти", программа завершает свою работу.

**Блок-схемы**

**Написание программы**

При написании программы была продумана структура данных и функции для управления списком.

Был реализован интерфейс пользователя с меню для удобства использования программы.

Особое внимание было уделено обработке граничных случаев, таких как вставка или удаление элементов в пустом списке или на несуществующих позициях.

Для каждой функции были написаны проверки на корректность входных данных.

Весь код был организован таким образом, чтобы обеспечить читаемость и легкость поддержки.

Эта программа может быть использована для управления данными студентов в учебных заведениях или других системах, где требуется хранение и обработка списков информации. Она демонстрирует использование односвязных списков и базовых операций над ними в языке программирования C.

**Примеры работы программы**

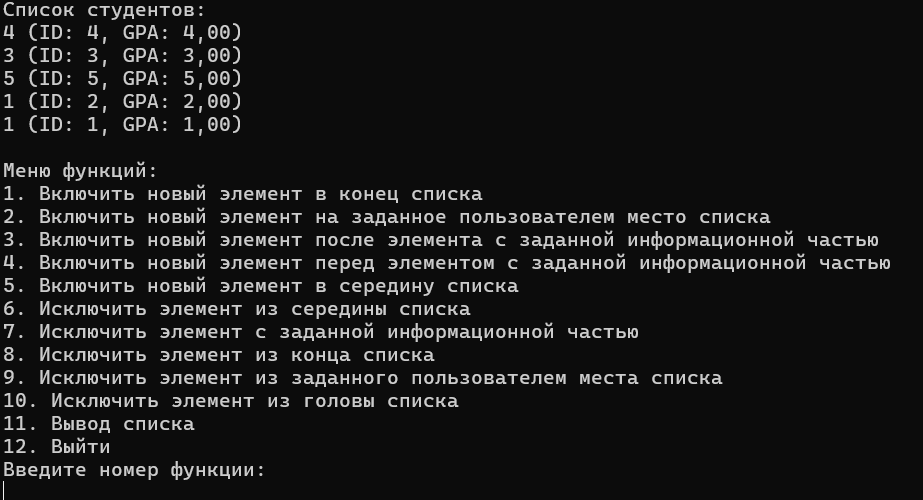


Рисунок 1 – Выполнения программы и вывод списка.

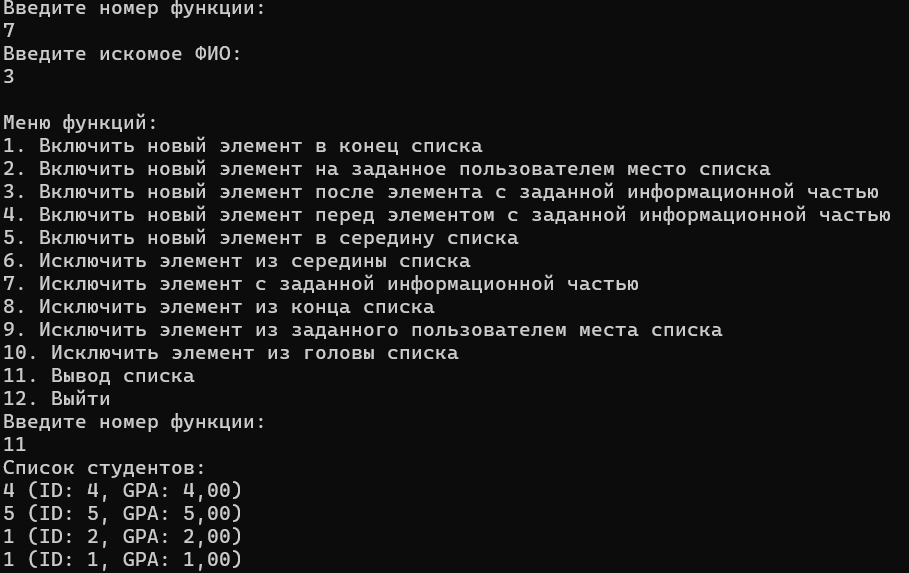


Рисунок 2 – Пример функции исключения элемента с заданной информационной частью.

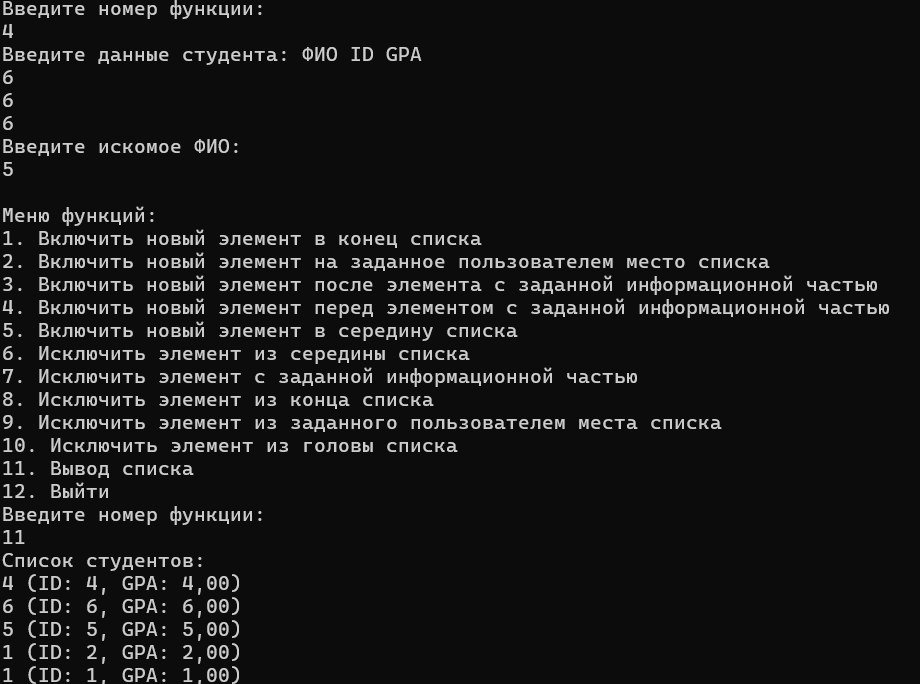


Рисунок 3 – Пример функции включения в список перед заданной информационной частью.

**Вывод**

# Освоили способы представления и обработки данных в виде списков.

# **Листинг (Приложение а)**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <iostream>

#pragma warning(disable : 4996)

#define MAX\_SIZE 50

// Структура для хранения информации о студенте

typedef struct student {

char name[50];

int student\_id;

float gpa;

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

} Student;

// Функция для создания нового элемента списка

Student\* createStudent(const char name[], int student\_id, float gpa) {

Student\* newStudent = (Student\*)malloc(sizeof(Student));

strcpy(newStudent->name, name);

newStudent->student\_id = student\_id;

newStudent->gpa = gpa;

newStudent->next = NULL;

return newStudent;

}

// Функция для вывода списка студентов

void printStudents(Student\* head) {

if (head == NULL) {

printf("Список студентов пуст\n");

}

else {

Student\* current = head;

while (current != NULL) {

printf("%s (ID: %d, GPA: %.2f)\n", current->name, current->student\_id, current->gpa);

current = current->next;

}

}

}

////////// Включение

// Функция для включения нового элемента в конец списка

void appendStudent(Student\*\* head, const char name[], int student\_id, float gpa) {

Student\* newStudent = createStudent(name, student\_id, gpa);

if (\*head == NULL) {

\*head = newStudent;

}

else {

Student\* current = \*head;

while (current->next != NULL) {

current = current->next;

}

current->next = newStudent;

}

}

// Функция для включения нового элемента на заданное пользователем место списка

void insertAtPosition(Student\*\* head, const char name[], int student\_id, float gpa, int position) {

if (position < 0) {

printf("Позиция должна быть неотрицательным числом.\n");

return;

}

Student\* newStudent = createStudent(name, student\_id, gpa);

if (position == 0) {

newStudent->next = \*head;

\*head = newStudent;

return;

}

Student\* current = \*head;

int currentPosition = 0;

while (current != NULL && currentPosition < position - 1) {

current = current->next;

currentPosition++;

}

if (current == NULL) {

printf("В списке меньше элементов, чем заданная позиция.\n");

return;

}

newStudent->next = current->next;

current->next = newStudent;

}

// Функция для включения нового элемента после элемента с заданной информационной частью

void insertAfter(Student\* head, const char targetName[], const char name[], int student\_id, float gpa) {

Student\* current = head;

while (current != NULL) {

if (strcmp(current->name, targetName) == 0) {

Student\* newStudent = createStudent(name, student\_id, gpa);

newStudent->next = current->next;

current->next = newStudent;

return;

}

current = current->next;

}

printf("Элемент с заданным именем не найден.\n");

}

// Функция для включения нового элемента перед элементом с заданной информационной частью

void insertBefore(Student\*\* head, const char targetName[], const char name[], int student\_id, float gpa) {

if (\*head == NULL) {

printf("Список пуст.\n");

return;

}

if (strcmp((\*head)->name, targetName) == 0) {

Student\* newStudent = createStudent(name, student\_id, gpa);

newStudent->next = \*head;

\*head = newStudent;

return;

}

Student\* current = \*head;

while (current->next != NULL) {

if (strcmp(current->next->name, targetName) == 0) {

Student\* newStudent = createStudent(name, student\_id, gpa);

newStudent->next = current->next;

current->next = newStudent;

return;

}

current = current->next;

}

printf("Элемент с заданным именем не найден.\n");

}

// Функция для включения нового элемента в середину списка

void insertInMiddle(Student\*\* head, const char name[], int student\_id, float gpa) {

if (\*head == NULL) {

printf("Список пуст.\n");

return;

}

Student\* slow = \*head;

Student\* fast = (\*head)->next;

while (fast != NULL && fast->next != NULL) {

slow = slow->next;

fast = fast->next->next;

}

Student\* newStudent = createStudent(name, student\_id, gpa);

newStudent->next = slow->next;

slow->next = newStudent;

}

////////// Удаление

// Функция для удаления элемента из конца списка

void deleteFromEnd(Student\*\* head) {

if (\*head == NULL) {

printf("Список пуст.\n");

return;

}

if ((\*head)->next == NULL) {

free(\*head);

\*head = NULL;

return;

}

Student\* current = \*head;

while (current->next->next != NULL) {

current = current->next;

}

free(current->next);

current->next = NULL;

}

// Функция для удаления элемента из заданного пользователем места списка

void deleteAtPosition(Student\*\* head, int position) {

if (\*head == NULL) {

printf("Список пуст.\n");

return;

}

if (position < 0) {

printf("Позиция должна быть неотрицательным числом.\n");

return;

}

if (position == 0) {

Student\* temp = \*head;

\*head = (\*head)->next;

free(temp);

return;

}

Student\* current = \*head;

int currentPosition = 0;

while (current != NULL && currentPosition < position - 1) {

current = current->next;

currentPosition++;

}

if (current == NULL || current->next == NULL) {

printf("В списке меньше элементов, чем заданная позиция.\n");

return;

}

Student\* temp = current->next;

current->next = current->next->next;

free(temp);

}

// Функция для удаления элемента из головы списка

void deleteFromHead(Student\*\* head) {

if (\*head == NULL) {

printf("Список пуст.\n");

return;

}

Student\* temp = \*head;

\*head = (\*head)->next;

free(temp);

}

// Функция для удаления элемента с заданной информационной частью

void deleteByInfo(Student\*\* head, const char targetName[]) {

if (\*head == NULL) {

printf("Список пуст.\n");

return;

}

if (strcmp((\*head)->name, targetName) == 0) {

Student\* temp = \*head;

\*head = (\*head)->next;

free(temp);

return;

}

Student\* current = \*head;

while (current->next != NULL) {

if (strcmp(current->next->name, targetName) == 0) {

Student\* temp = current->next;

current->next = current->next->next;

free(temp);

return;

}

current = current->next;

}

printf("Элемент с заданным именем не найден.\n");

}

// Функция для удаления элемента из середины списка

void deleteFromMiddle(Student\*\* head) {

if (\*head == NULL || (\*head)->next == NULL) {

printf("Список пуст или содержит только один элемент.\n");

return;

}

Student\* slowPtr = \*head; // Медленный указатель

Student\* fastPtr = \*head; // Быстрый указатель

Student\* prevPtr = NULL; // Указатель на предыдущий узел перед серединой

while (fastPtr != NULL && fastPtr->next != NULL) {

fastPtr = fastPtr->next->next; // Быстрый указатель двигается на два узла вперед

prevPtr = slowPtr; // Сохраняем предыдущее значение медленного указателя

slowPtr = slowPtr->next; // Медленный указатель двигается на один узел вперед

}

// Удаление узла, на который указывает медленный указатель (это узел в середине списка)

if (prevPtr == NULL) { // Если prevPtr остался NULL, значит удаляемый элемент - голова списка

\*head = (\*head)->next;

}

else {

prevPtr->next = slowPtr->next;

}

free(slowPtr);

}

/////////

// Вывод пунктов меню функций

int menu\_func() {

int choise;

puts("\nМеню функций: ");

puts("1. Включить новый элемент в конец списка");

puts("2. Включить новый элемент на заданное пользователем место списка");

puts("3. Включить новый элемент после элемента с заданной информационной частью");

puts("4. Включить новый элемент перед элементом с заданной информационной частью");

puts("5. Включить новый элемент в середину списка");

puts("6. Исключить элемент из середины списка");

puts("7. Исключить элемент с заданной информационной частью");

puts("8. Исключить элемент из конца списка");

puts("9. Исключить элемент из заданного пользователем места списка");

puts("10. Исключить элемент из головы списка");

puts("11. Вывод списка");

puts("12. Выйти");

puts("Введите номер функции: ");

scanf("%i", &choise);

return choise;

}

///////

int main() {

setlocale(LC\_ALL, "Russian");

Student\* head = NULL; // Инициализация списка студентов

// Добавление студентов в конец списка

appendStudent(&head, "Иванов Иван", 123, 3.5);

appendStudent(&head, "Петров Петр", 456, 4.0);

appendStudent(&head, "Сидоров Сидор", 789, 3.7);

printStudents(head);

// Переменные для меню

int choise = menu\_func();

char name[MAX\_SIZE];

int student\_id;

float gpa;

int position;

char targetName[MAX\_SIZE];

// Меню функций

while (choise != 12) {

if (choise < 6) {

puts("Введите данные студента: ФИО ID GPA");

scanf("%s", &name);

scanf("%i", &student\_id);

scanf("%f", &gpa);

}

if (choise == 3 || choise == 4 || choise == 7) {

puts("Введите искомое ФИО:");

scanf("%s", &targetName);

}

if (choise == 2 || choise == 9) {

puts("Введите искомое место:");

scanf("%i", &position);

position--;

}

switch (choise) {

case 1:

// Включение нового элемента в конец списка

appendStudent(&head, name, student\_id, gpa);

break;

case 2:

// Включение нового элемента на заданное пользователем место списка

insertAtPosition(&head, name, student\_id, gpa, position);

break;

case 3:

// Включение нового элемента после элемента с заданной информационной частью

insertAfter(head, targetName, name, student\_id, gpa);

break;

case 4:

// Включение нового элемента перед элементом с заданной информационной частью

insertBefore(&head, targetName, name, student\_id, gpa);

break;

case 5:

// Включение нового элемента в середину списка

insertInMiddle(&head, name, student\_id, gpa);

break;

case 6:

deleteFromMiddle(&head);

break;

case 7:

// Удаление элемента с заданной информационной частью

deleteByInfo(&head, targetName);

break;

case 8:

// Удаление элемента из конца списка

deleteFromEnd(&head);

break;

case 9:

// Удаление элемента из заданного пользователем места списка

deleteAtPosition(&head, position);

break;

case 10:

// Удаление элемента из головы списка

deleteFromHead(&head);

break;

case 11:

// Вывод списка студентов

printf("Список студентов:\n");

printStudents(head);

break;

case 12:

return 0;

}

choise = menu\_func();

}

}