**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра Вычислительной техники**

**Курсовая работа**

**по дисциплине «Программирование»**

**Тема: Разработка электронной картотеки**

|  |  |  |
| --- | --- | --- |
| Студент гр. 3312 |  | Шарапов И. Д. |
| Преподаватель |  | Аббас С. А. |

Санкт-Петербург

2024

**Содержание**

[Цель работы 3](#_Toc168001981)

[Задание 3](#_Toc168001982)

[Постановка задачи и описание решения 4](#_Toc168001983)

[Описание структур 6](#_Toc168001984)

[Описание функций 6](#_Toc168001985)

[Структура вызова функций 8](#_Toc168001986)

[Описание переменных 8](#_Toc168001987)

[Схема алгоритма 12](#_Toc168001988)

[Текст программы 23](#_Toc168001989)

[Контрольные примеры 34](#_Toc168001990)

[Содержимое файлов 34](#_Toc168001991)

[Примеры выполнения программы 34](#_Toc168001992)

[Выводы 34](#_Toc168001993)

# Цель работы

Полное решение содержательной задачи (содержательная и формальная постановка задачи, спецификация, включая описание диалога, выбор метода решения и структур данных, разработка алгоритма, программная реализация, тестирование и отладка, документирование). Создание электронной картотеки спортсменов.

# Задание

Создать электронную картотеку, хранящуюся на диске, и программу на языке Си, обеспечивающую взаимодействие с ней. Программа выполняет:

* занесение данных в электронную картотеку;
* внесение изменений (исключение, корректировка, добавление);
* поиск данных по различным признакам;
* сортировку по различным признакам;
* вывод результатов на экран и сохранение на диске.

Выбор подлежащих выполнению команд должен быть реализован с помощью основного меню и вложенных меню.

Задача должна быть структурирована и отдельные части должны быть оформлены как функции.

Исходные данные должны вводиться с клавиатуры. В процессе обработки картотека должна храниться в памяти компьютера в виде списков и массивов структур, связанных указателями. Типы списков и структур выбираются исходя из предметной области.

Картотека составляется по выбранной предметной области.

В программе должно быть реализовано простейшее меню. Выполнение программы должно быть многократным по желанию пользователя. Данные первоначально считываются из файла (файлов), в процессе работы данные вводятся с клавиатуры.

Перечень пунктов меню:

1. Вывод картотеки. (В консоль в виде таблички);
2. Поиск карточек по параметру (Выбор параметра и последующий поиск по строке);
3. Сортировка картотеки по параметру;
4. Добавление карточки спортсмена (Ввод строки в формате);
5. Редактирование карточки по ID (Последовательный ввод полей, либо пропуск);
6. Удаление карточек по параметру (Поиск подходящих строк и подтверждение их удаления);
7. Сохранение на диск;
8. Выход из программы.

# Постановка задачи и описание решения

Для решения задачи необходимо написать программу, которая взаимодействует с электронной картотекой. Для этого используется структура Athlete, в которой содержится 7 полей: имя спортсмена; университет, в котором учится спортсмен; возраст; вес; рост; массив из трёх чисел (результаты выступления спортсмена); индекс результатов спортсмена (отношение суммы к весу).

Вначале программа спрашивает у пользователя из какого файла взять первичную информацию. Далее выводится главное меню (список всех команд, доступных пользователю), для этого программа обращается к функции *help()*. Список команд, которые она выводит:

*!print* – вывод всех карточек в виде таблице. Программа обращается к функции *print()*.

*!find* – поиск карточек по параметру. Программа вызывает функцию *find()*. Пользователю предлагается выбрать один из 10 параметров для сортировки, либо 0 для выхода в главное меню. Если пользователь выбрал не 0, то у него спрашивается подстрока, которая будет искаться без учёта регистра. Пользователю выводится все найденные пользователи, либо сообщение *«No matches found!»*. В первом случае пользователю предлагается сортировать найденные данные по всем параметрам, при этом исходный список не будет меняться. Для этого используется функция *sorted()*.

*!sort* - сортировка карточек по параметру. Программа вызывает функцию *sort()*. Пользователю предлагается сортировать данные по любому параметру, до тех пор, пока он не введёт 0. При этом данные в памяти компьютера тоже меняют своё положение.

*!add* – добавление карточки. Программа вызывает функцию *add()*. Пользователю выводится формат, в котором он должен ввести данные о новом спортсмене, далее спрашивается сама строка. Программа обрабатывает данную строку и добавляет спортсмена в конец списка.

*!edit* – изменение конкретной карточки. Программа вызывает функцию *edit()*. Далее у пользователя спрашивается ID спортсмена, которого он хочет изменить. Если спортсмен с данным ID существует, пользователю предлагается последовательно менять поля этого спортсмена. Если пользователь не хочет менять конкретное поле, он может нажать *ENTER*.

*!delete* – удаление карточек по параметру. Программа вызывает функцию *delete()*. Как и в *find()* выбираются спортсмены и выводятся пользователю. Далее у пользователя спрашивается подтверждение: точно ли он хочет удалить их. Если пользователь ответит *Y*, то данные об этих пользователях удаляются, а память очищается. В конце пользователю выводится изменённый список.

*!save* – сохранение данных в файл. Программа вызывает функцию *save()*. У пользователя спрашивается имя файла, в который он хочет сохранить данные. После считывания строки данные сохраняются в этот файл в формате CSV.

*!end* – завершает выполнение программы и очищает память.

При переходе из главного меню к конкретному функционалу экран очищается, и пользователю для наглядности выводится вся табличка. В конце выполнения функции экран также очищается, и пользователь «попадает» в главное меню, где уже сразу отображён список всех возможных команд.

# Описание структур

1. Структура Athlete

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Имя переменной** | **Тип** | **Назначение** |
| 1 | name | char\* | Имя спортсмена |
| 2 | university | char\* | Название университета |
| 3 | age | int | Возраст |
| 4 | weight | float | Вес в килограммах |
| 5 | height | int | Рост в сантиметрах |
| 6 | result | int[3] | Результаты выступления |
| 7 | index | float | Отношение результата к весу |

2. Структура NodeOfList

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Имя переменной** | **Тип** | **Назначение** |
| 1 | id | int | Уникальный ID |
| 2 | data | Athlete\* | Указатель на данные о спортсмене |
| 3 | next | struct NodeOfList\* | Указатель на следующую вершину списка |
| 4 | prev | struct NodeOfList\* | Указатель на предыдущую вершину списка |

3. Структура ListOfAthlete

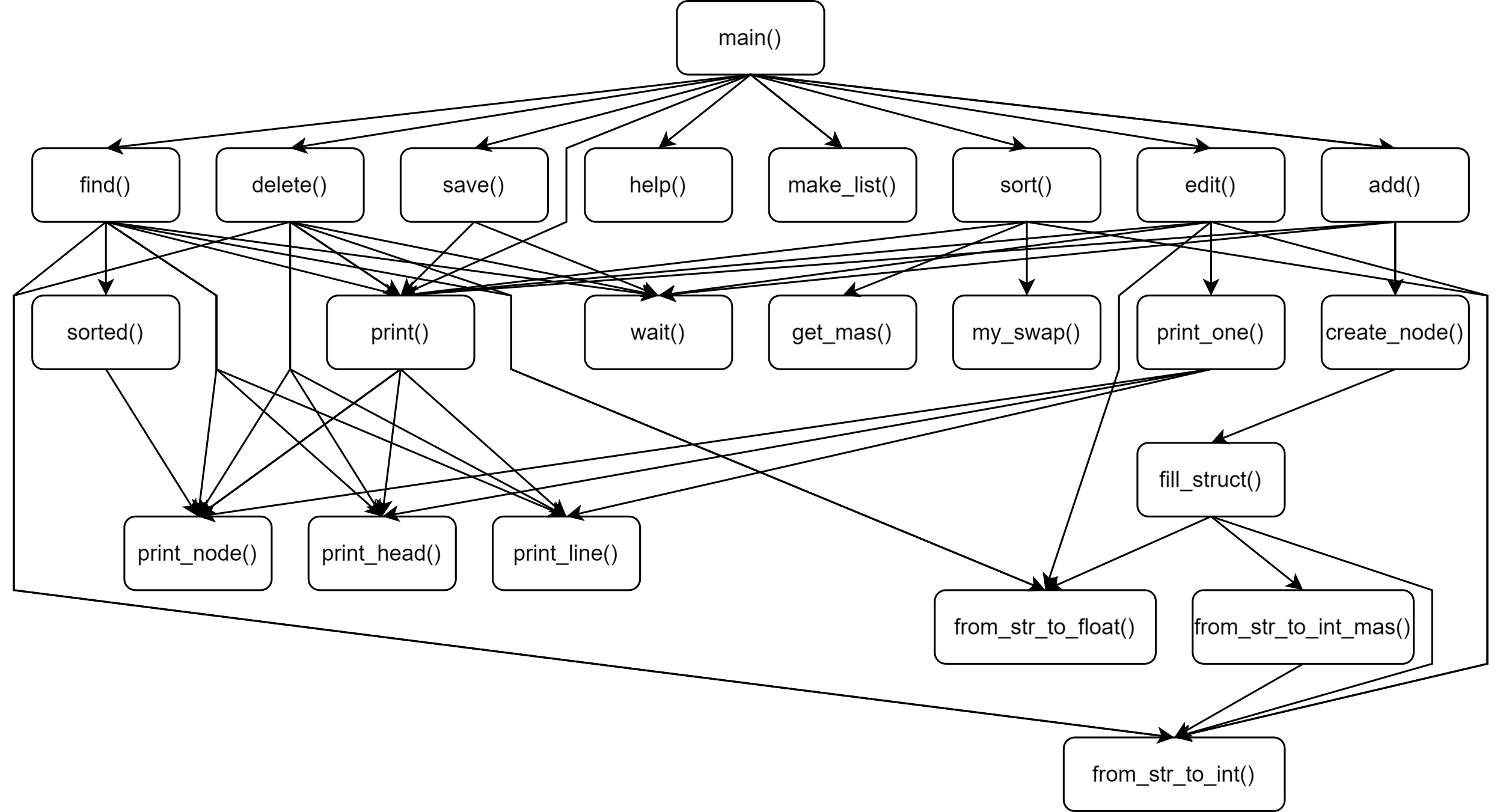
|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Имя переменной** | **Тип** | **Назначение** |
| 1 | length | int | Уникальный ID |
| 2 | first | struct NodeOfList\* | Указатель на первую вершину списка |
| 3 | last | struct NodeOfList\* | Указатель на последнюю вершину списка |

# Описание функций

|  |  |  |
| --- | --- | --- |
| **№** | **Название** | **Назначение** |
| 1 | main | Основная функция программы. Открывает файл, инициализирует список, отвечает за взаимодействие с пользователем через меню. В конце очищает память. |
| 2 | from\_str\_to\_int | Конвертирует строку в целочисленное значение. Если символ не цифра – возвращает 0. |
| 3 | from\_str\_to\_float | Конвертирует строку в число с плавающей точкой. Если символ не цифра – возвращает 0. |
| 4 | from\_str\_to\_int\_mas | Конвертирует строку в массив целочисленных значений длины 3. |
| 5 | fill\_struct | Извлекает данные из строки и заполняет поля структуры Athlete. |
| 6 | make\_list | Создаёт список (выделяет память и возвращает указатель на него). |
| 7 | create\_node | Выделяет память под новую вершину списка, заполняет его данными и возвращает указатель на него. |
| 8 | help | Отображает доступные команды пользователю в консоли. |
| 9 | wait | Приостанавливает выполнение программы до нажатия клавиши Enter. |
| 10 | print\_line | Выводит линию для разделения таблицы данных в консоли. |
| 11 | print\_head | Выводит поля заголовка таблицы в консоль. |
| 12 | print\_node | Выводит данные одного элемента списка в консоль. |
| 13 | print\_one | Выводит данные одного элемента списка с заголовком в консоль. |
| 14 | print | Выводит все элементы списка в консоль. |
| 15 | sorted | Сортировка выбранных элементов списка и вывод отсортированных данных в консоль. (не влияет на порядок основного списка) |
| 16 | find | Поиск элементов списка и вывод найденных данных в консоль с возможностью последующей сортировки этих данных. |
| 17 | get\_mas | Функция проходит по всему списку и заполняет массив указателями на вершины списка. Возвращает массив указателей. |
| 18 | my\_swap | Меняет два значения в списке местами. |
| 19 | sort | Сортирует элементы списка в соответствии с заданным параметром и выводит отсортированные данные в консоль. |
| 20 | add | Пользователь вводит данные нового элемента, функция создаёт новою вершину и добавляет её в конец списка. |
| 21 | edit | Пользователь вводит ID элемента, который хочет отредактировать, и новые данные. Функция находит элемент по ID и предлагает изменить его данные по отдельным полям. |
| 22 | delete | Пользователь выбирает поле для удаления и вводит критерии поиска. Функция удаляет элементы, соответствующие заданным критериям. |

|  |  |  |
| --- | --- | --- |
| 23 | save | Пользователь выбирает имя файла, в который будут сохранены данные списка. Функция записывает все данные в файл в соответствии с форматом. |

# Структура вызова функций



# Описание переменных

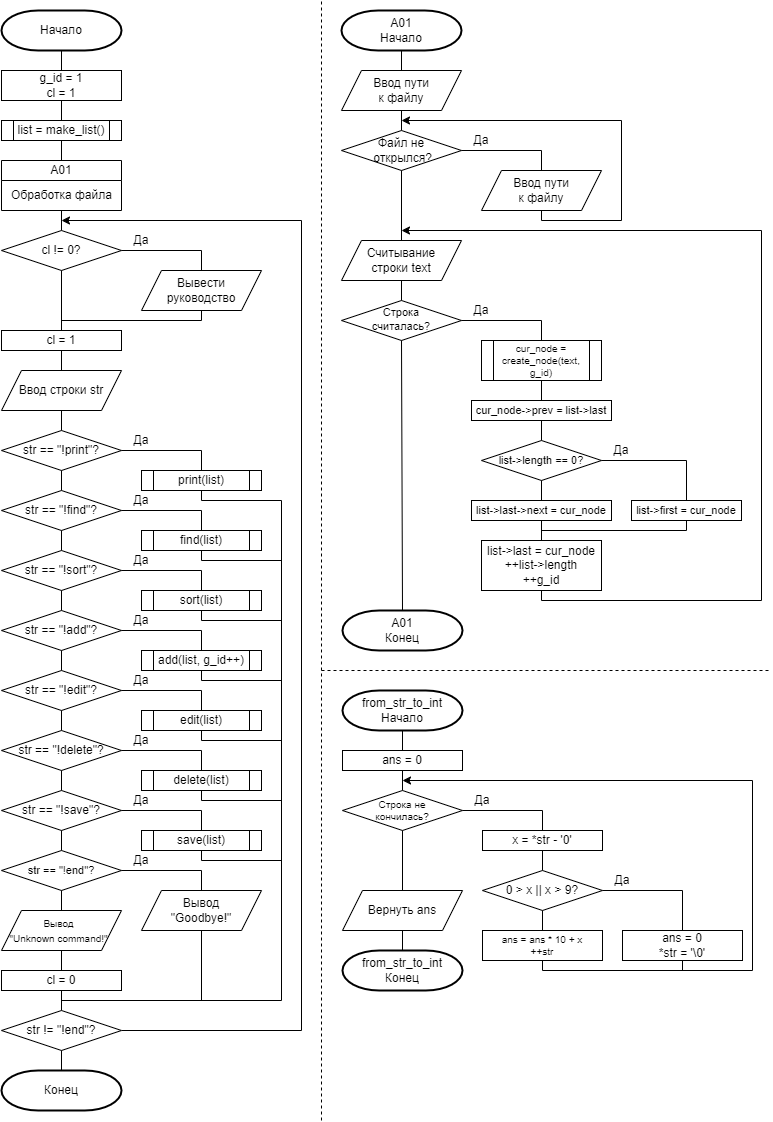
|  |  |  |  |
| --- | --- | --- | --- |
| Функция *int main()* | | | |
| № | Имя переменной | Тип | Назначение |
| 1 | list | ListOfAthlete\* | Список спортсменов |
| 2 | g\_id | int | Глобальный ID |
| 3 | cl | int | Флаг для отображения help |
| 4 | filename | char[] | Буфер для хранения имени файла |
| 5 | str | char[] | Буфер для команд пользователя |
| 6 | text | char[] | Буфер для содержимого файла |
| 7 | cur\_node | NodeOfList\* | Текущий элемент в списке |
| 8 | f | FILE\* | Указатель на файл |
| Функция *int from\_str\_to\_int(char \*str)* | | | |
| 1 | str | char\* | Стока, которую нужно конвертировать в число |
| 2 | ans | int | Результирующее число |
| 3 | x | int | Текущая цифра |

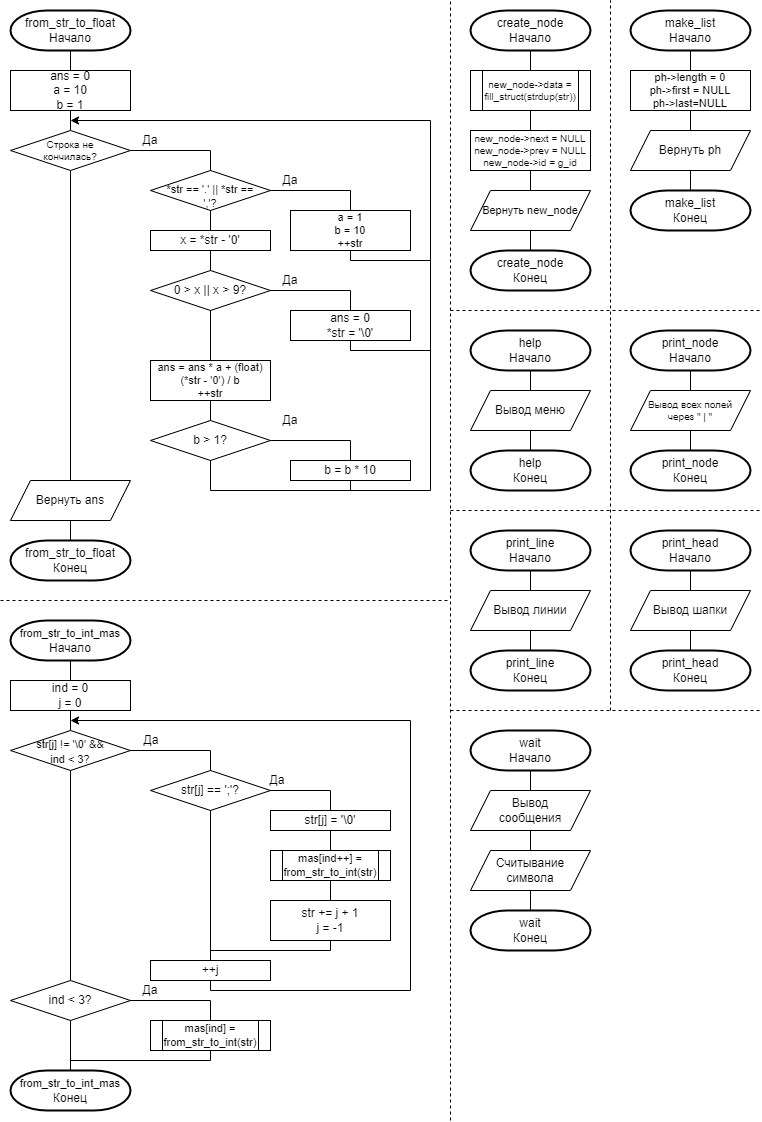
|  |  |  |  |
| --- | --- | --- | --- |
| Функция *float from\_str\_to\_float(char \*str)* | | | |
| 1 | str | char\* | Стока, которую нужно конвертировать в число с плавающей точкой |
| 2 | ans | float | Результирующее число с плавающей точкой |
| 3 | a | float | Целая часть числа |
| 4 | b | float | Дробная часть числа |
| 5 | x | float | Текущая цифра |
| Функция *void from\_str\_to\_int\_mas(char \*str, int \*mas)* | | | |
| 1 | str | char\* | Стока, которую нужно конвертировать в массив чисел длины 3 |
| 2 | mas | int\* | Указатель на массив, в который будут записаны числа |
| 3 | ind | int | Текущий индекс в массиве |
| 4 | j | int | Текущий индекс в строке |
| Функция *Athlete \*fill\_struct(char \*str)* | | | |
| 1 | str | char\* | Строка, которую нужно конвертировать в сущность Athlete |
| 2 | user | Athlete\* | Сущность Athlete |
| 3 | word | char\* | Текущая подстрока в строке |
| 4 | pole | char\*[] | Массив указателей на подстроки |
| 5 | ind | int | Текущий индекс в массиве подстрок |
| 6 | tt | int | Текущий индекс в строке |
| Функция *ListOfAthlete \*make\_list()* | | | |
| 1 | ph | ListOfAthlete\* | Указатель на создаваемый список |
| Функция *NodeOfList \*create\_node(char \*str, int g\_id)* | | | |
| 1 | str | char\* | Строка, которую нужно конвертировать в сущность Athlete |
| 2 | g\_id | int | Глобальный ID |
| 3 | new\_node | NodeOfList\* | Указатель на создаваемую вершину |
| Функция *void print\_node(NodeOfList \*node)* | | | |
| 1 | node | NodeOfList\* | Вершина списка, которую нужно вывести |
| Функция *void print\_one(NodeOfList \*node)* | | | |
| 1 | node | NodeOfList\* | Вершина списка, которую нужно вывести |
| Функция *void print(ListOfAthlete \*list)* | | | |
| 1 | list | ListOfAthlete\* | Список, который нужно вывести |
| Функция *void sorted(int \*mas, ListOfAthlete \*list, int param)* | | | |
| 1 | mas | int\* | Массив флагов для сортировки |
| 2 | list | ListOfAthlete\* | Список спортсменов |
| 3 | param | int | Параметр, по которому сортируем |
| 4 | cur\_node | NodeOfList\* | Текущая вершина списка |
| 5 | min\_node | NodeOfList\* | Минимальная вершина списка |
| 6 | ind | int | Индекс минимальной вершины |

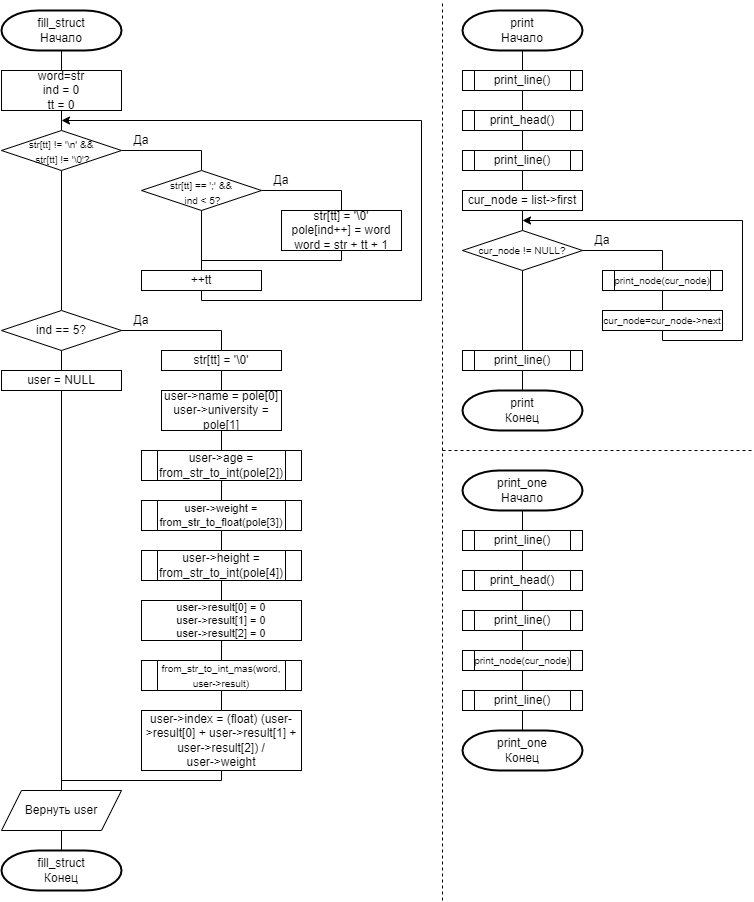
|  |  |  |  |
| --- | --- | --- | --- |
| Функция *void find(ListOfAthlete \*list)* | | | |
| 1 | list | ListOfAthlete\* | Список спортсменов |
| 2 | cur\_node | NodeOfList\* | Текущая вершина списка |
| 3 | str | char[] | Строка, которую ищет пользователь |
| 4 | mas | int[] | Массив флагов для поиска |
| 5 | fl | int | Флаг на то, что хоть один элемент найден |
| 6 | param | int | Параметр, по которому ищем |
| Функция *NodeOfList \*\*get\_mas(ListOfAthlete \*list)* | | | |
| 1 | list | ListOfAthlete\* | Список спортсменов |
| 2 | cur\_node | NodeOfList\* | Текущая вершина списка |
| 3 | mas | NodeOfList\*\* | Массив указателей на вершины списка |
| Функция *void my\_swap(NodeOfList \*\*mas, ListOfAthlete \*list, int i, int j)* | | | |
| 1 | mas | NodeOfList\*\* | Массив указателей на вершины списка |
| 2 | list | ListOfAthlete\* | Список спортсменов |
| 3 | i | int | Индекс первого узла |
| 4 | j | int | Индекс второго узла |
| 5 | q | NodeOfList\* | Вспомогательная вершина |
| Функция *void sort(ListOfAthlete \*list)* | | | |
| 1 | list | ListOfAthlete\* | Список спортсменов |
| 2 | mas | NodeOfList\*\* | Массив указателей на вершины списка |
| 3 | str | char[] | Параметр в виде строки |
| 4 | n | int | Количество вершин в списке |
| 5 | param | int | Параметр, по которому сортируем |
| Функция *void add(ListOfAthlete \*list, int g\_id)* | | | |
| 1 | list | ListOfAthlete\* | Список спортсменов |
| 2 | g\_id | int | Глобальный ID |
| 3 | str | char[] | Буфер для данных о спортсмене |
| 4 | cur\_node | NodeOfList\* | Текущая вершина списка |
| Функция *void edit(ListOfAthlete \*list)* | | | |
| 1 | list | ListOfAthlete\* | Список спортсменов |
| 2 | the\_node | NodeOfList\* | Вершина, которую редактируем |
| 3 | id | int | ID спортсмена для редактирования |
| 4 | str | char[] | Буфер для пользовательского ввода |
| 5 | s\_id | char[] | Буфер ID спортсмена в строковом виде |

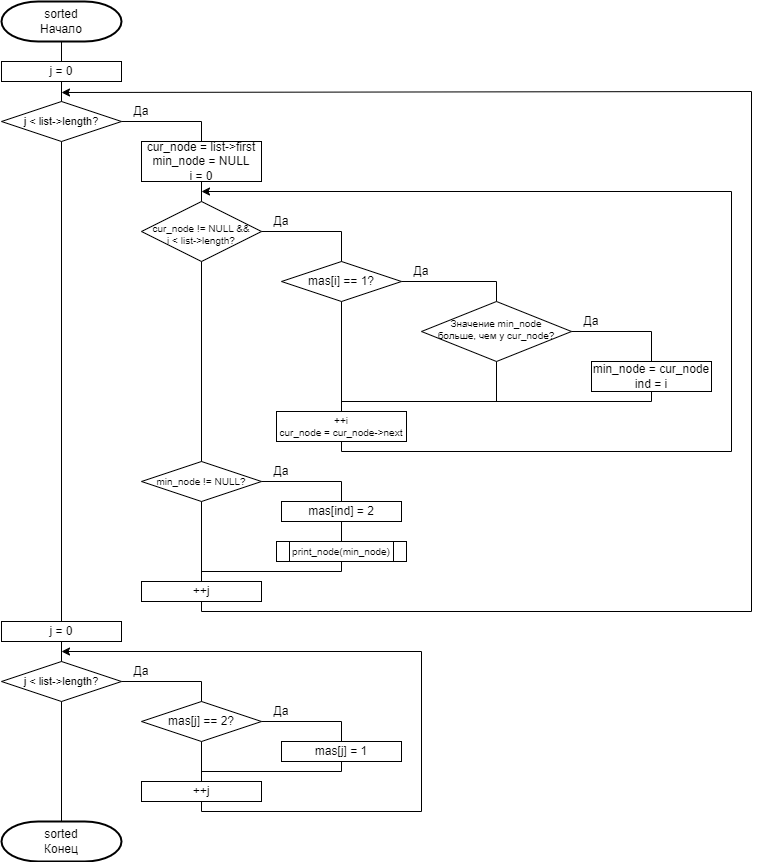
|  |  |  |  |
| --- | --- | --- | --- |
| Функция *void delete(ListOfAthlete \*list)* | | | |
| 1 | list | ListOfAthlete\* | Список спортсменов |
| 2 | cur\_node | NodeOfList\* | Текущая вершина списка |
| 3 | prev\_node | NodeOfList\* | Предыдущая вершина списка |
| 4 | str | char[] | Строка, которую ищет пользователь |
| 5 | ch | char | Символ для подтверждения |
| 6 | mas | int[] | Массив флагов для удаления |
| 7 | fl | int | Флаг на то, что хоть один элемент найден |
| 8 | param | int | Параметр, по которому ищем |
| 9 | cnt | int | Количество элементов для удаления |
| Функция *void save(ListOfAthlete \*list)* | | | |
| 1 | list | ListOfAthlete\* | Список спортсменов |
| 2 | f | FILE\* | Указатель на файл |
| 3 | filename | char[] | Буфер для имени файла |

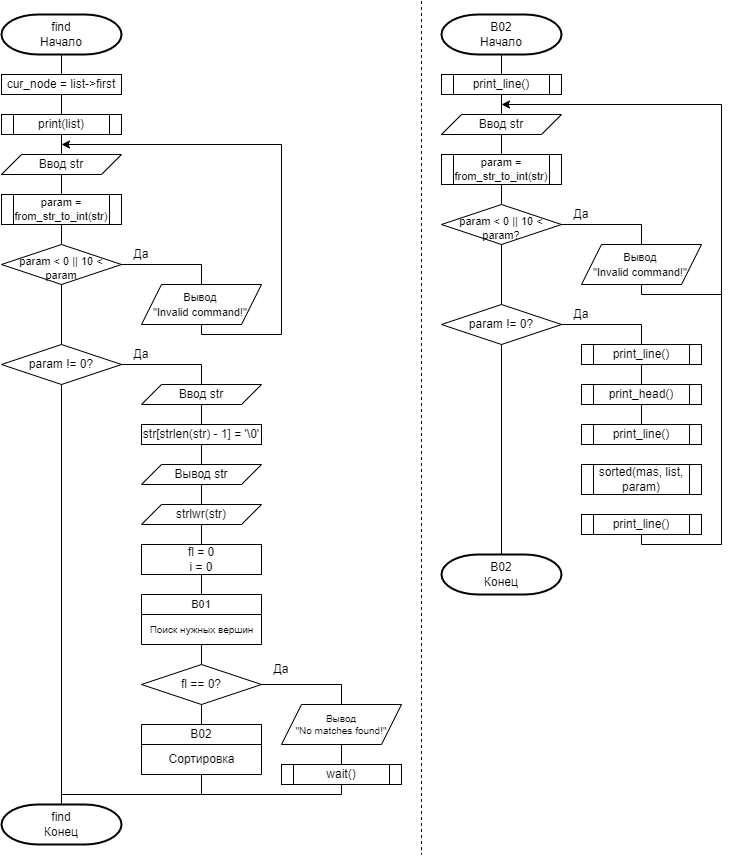
# **Схема алгоритма**

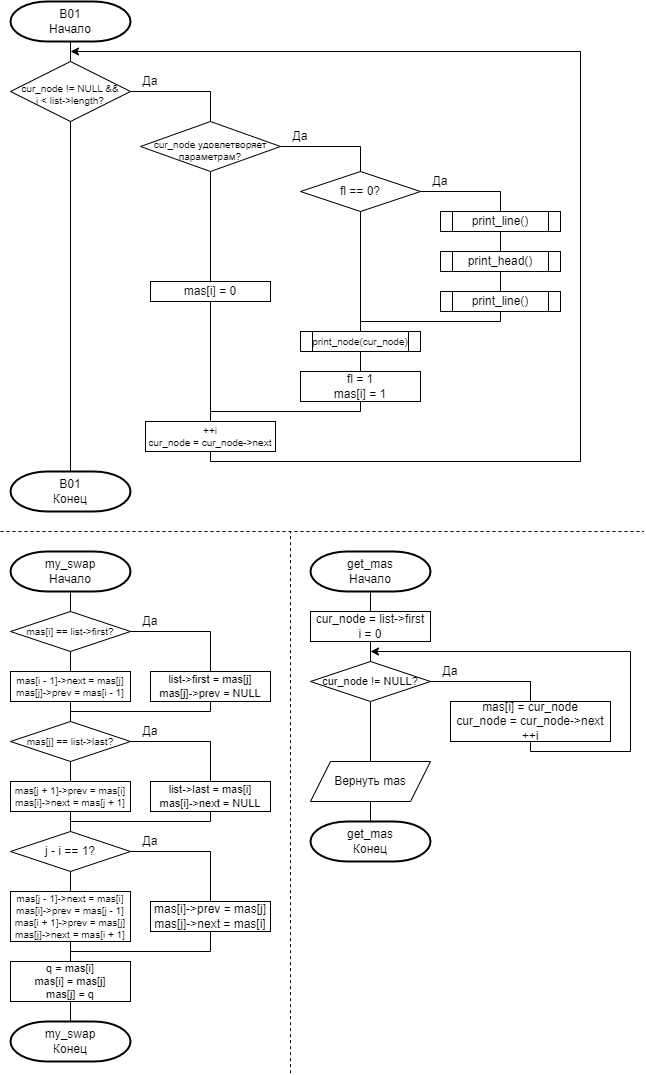


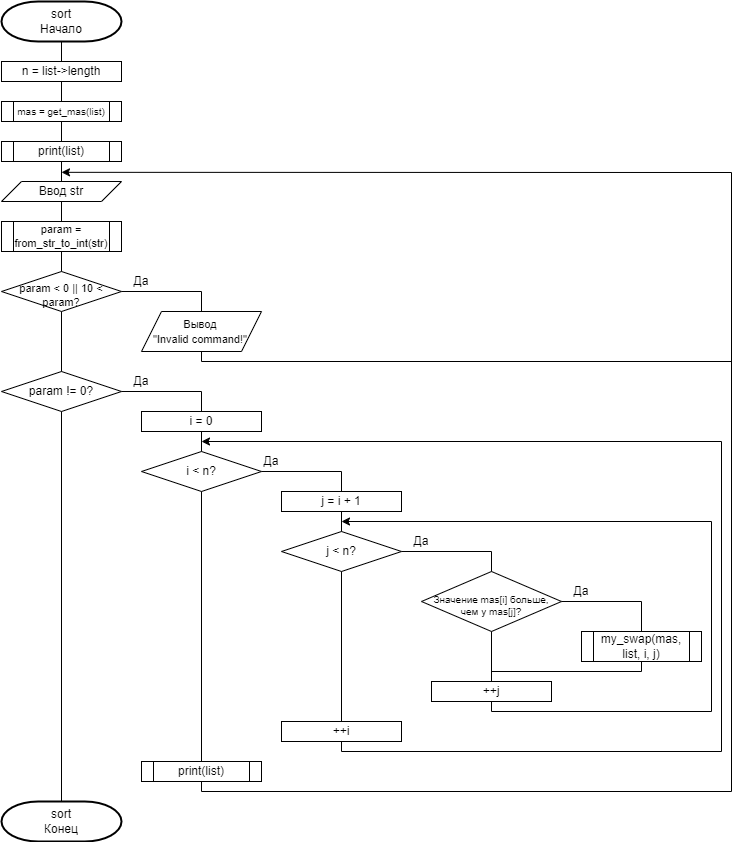


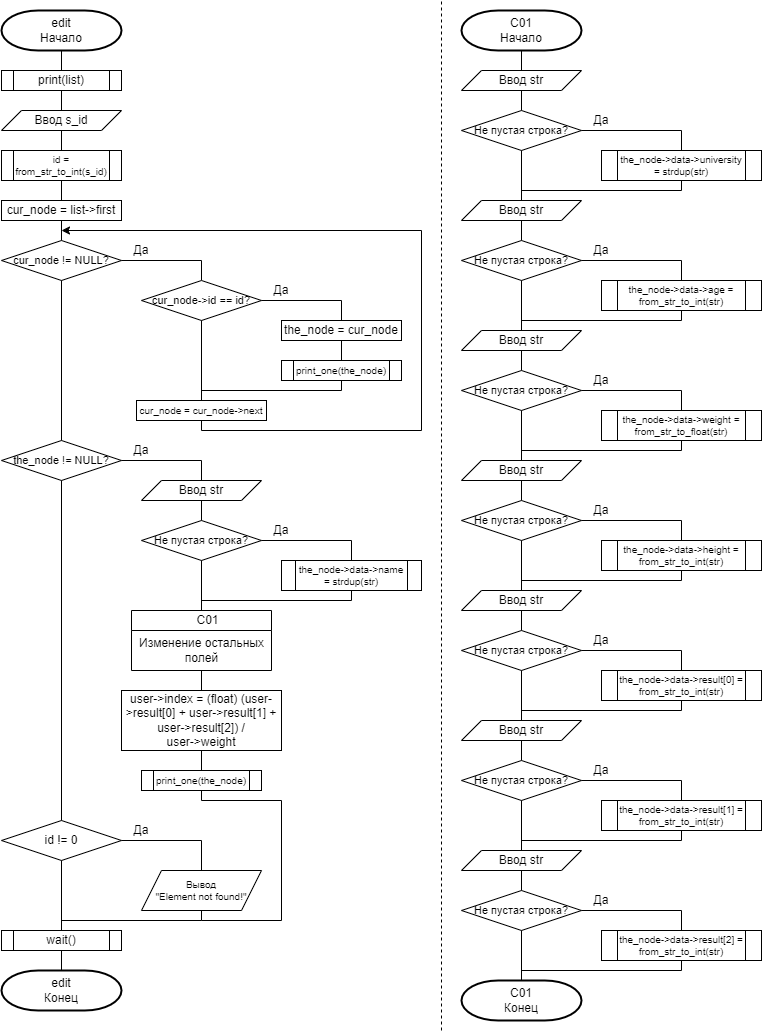


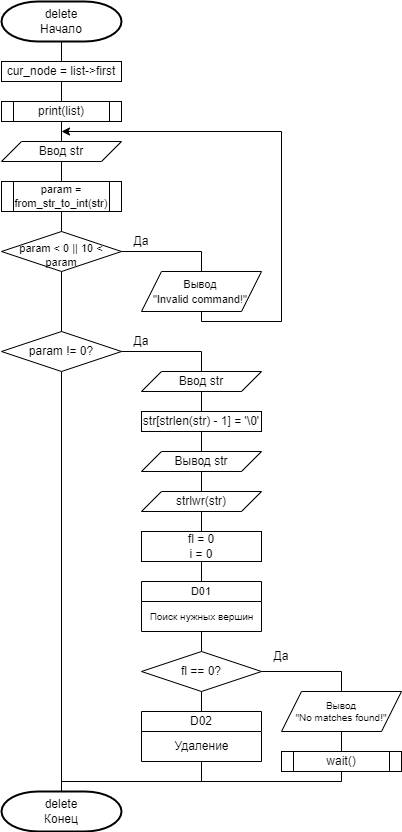


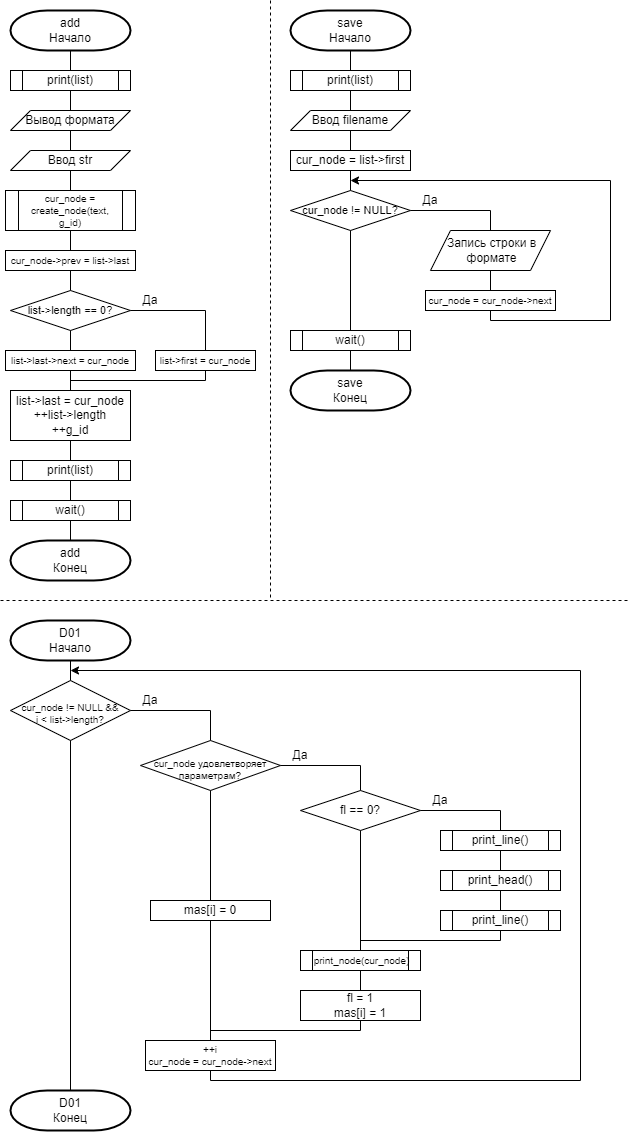


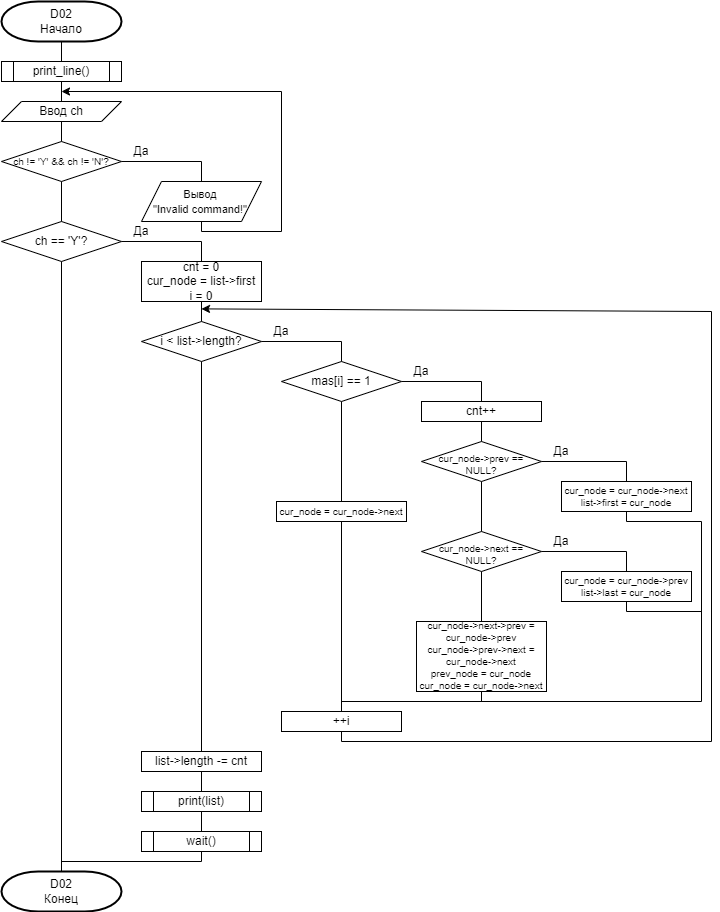












# Текст программы

|  |
| --- |
| **#include <stdio.h> #include <stdlib.h> #include <string.h> #include <math.h>  #ifdef WIN32 #define CLS system("cls") #else #define CLS system("clear") #endif   typedef struct Athlete {  char \*name; /\* Pointer to the athlete's name \*/  char \*university; /\* Pointer to the athlete's university name \*/  int age; /\* Athlete's age \*/  float weight; /\* Athlete's weight in kilograms \*/  int height; /\* Athlete's height in centimeters \*/  int result[3]; /\* Array to store results of the athlete's performance \*/  float index; /\* Index value calculated based on the athlete's performance \*/ } Athlete;  typedef struct NodeOfList {  int id; /\* Unique ID for the node \*/  Athlete \*data; /\* Pointer to the data (athlete information) stored in the node \*/  struct NodeOfList \*next; /\* Pointer to the next node in the list \*/  struct NodeOfList \*prev; /\* Pointer to the previous node in the list \*/ } NodeOfList;  typedef struct ListOfAthlete {  int length; /\* Number of nodes (athletes) in the list \*/  NodeOfList \*first; /\* Pointer to the first node in the list \*/  NodeOfList \*last; /\* Pointer to the last node in the list \*/ } ListOfAthlete;   /\* Convert a string to an integer \*/ int from\_str\_to\_int(char \*str);  /\* Convert a string to a floating-point number \*/ float from\_str\_to\_float(char \*str);  /\* Convert a string containing delimited integers to an integer array \*/ void from\_str\_to\_int\_mas(char \*str, int \*mas);  /\* Fill an Athlete structure with data from a string \*/ Athlete \*fill\_struct(char \*str);  /\* Create an empty list of athletes \*/ ListOfAthlete \*make\_list();  /\* Create a new node for the list \*/ NodeOfList \*create\_node(char \*str, int g\_id);  /\* Display the list of available commands \*/ void help();  /\* Wait for user input to continue \*/ void wait();  /\* Print a line separator \*/ void print\_line();  /\* Print the header with column names \*/ void print\_head();  /\* Print data of a single node \*/ void print\_node(NodeOfList \*node);  /\* Print data of a single node with header and footer \*/ void print\_one(NodeOfList \*node);  /\* Print data of all nodes in the list \*/ void print(ListOfAthlete \*list);  /\* Sort and print nodes based on a specified parameter \*/ void sorted(int \*mas, ListOfAthlete \*list, int param);  /\* Find and print nodes based on user input \*/ void find(ListOfAthlete \*list);  /\* Creates an array of pointers to NodeOfList structures based on the given list \*/ NodeOfList \*\*get\_mas(ListOfAthlete \*list);  /\* Swaps two nodes in the list and updates their positions \*/ void my\_swap(NodeOfList \*\*mas, ListOfAthlete \*list, int i, int j);  /\* Sorts the list of athletes based on a selected parameter \*/ void sort(ListOfAthlete \*list);  /\* Adds a new athlete to the list \*/ void add(ListOfAthlete \*list, int g\_id);  /\* Allows editing the details of an athlete in the list based on the provided ID \*/ void edit(ListOfAthlete \*list);  /\* Deleting athletes from the list based on the specified parameter \*/ void delete(ListOfAthlete \*list);  /\* Save the data of athletes stored in a linked list to a file specified by the user \*/ void save(ListOfAthlete \*list);  int main() {  ListOfAthlete \*list; /\* Pointer to the list of athletes \*/  int g\_id, cl; /\* Variables for athlete ID and command line flag \*/  char filename[128], str[128], text[1024]; /\* Buffers for filename, user input, and file content \*/  NodeOfList \*cur\_node = NULL; /\* Pointer to the current node in the list \*/  FILE \*f; /\* File pointer \*/   g\_id = 1; /\* Initialize athlete ID \*/  cl = 1; /\* Initialize command line input flag \*/  list = make\_list(); /\* Create an empty list of athletes \*/  printf("Please enter the file name:\n");  fgets(filename, sizeof(filename), stdin); /\* Read the filename from the user \*/  filename[strcspn(filename, "\n")] = '\0'; /\* Remove the newline character from the input \*/  f = fopen(filename, "r"); /\* Open the file for reading \*/  while (f == NULL) { /\* Loop until a valid file is opened \*/  printf("Something went wrong!\n"  "Perhaps such a file does not exist.\n"  "Please enter the file name again:\n");  fgets(filename, sizeof(filename), stdin); /\* Read the filename again if opening fails \*/  filename[strcspn(filename, "\n")] = '\0'; /\* Remove the newline character from the input \*/  f = fopen(filename, "r"); /\* Attempt to open the file again \*/  }   while (fgets(text, sizeof(text), f)) { /\* Read each line from the file \*/  cur\_node = create\_node(text, g\_id); /\* Create a new node with the text and assign a unique ID \*/  if (cur\_node != NULL) {  cur\_node->prev = list->last; /\* Set the previous node pointer \*/  if (list->length == 0) {  list->first = cur\_node; /\* Set the first node if the list is empty \*/  } else {  list->last->next = cur\_node; /\* Link the new node to the end of the list \*/  }  list->last = cur\_node; /\* Update the last node pointer \*/  ++list->length; /\* Increment the list length \*/  ++g\_id;  }  }   CLS; /\* Clear the screen \*/  printf("The file has been successfully processed!\n");  fclose(f); /\* Close the file \*/   do {  if (cl) help(); /\* Display help information if the flag is set \*/  cl = 1; /\* Reset the flag \*/  fgets(str, sizeof(str), stdin); /\* Read a command from the user \*/  str[strcspn(str, "\n")] = '\0'; /\* Remove the newline character from the input \*/  if (!strcmp(str, "!print")) { /\* Compare the command with "!print" \*/  CLS;  print(list); /\* Print the list of athletes \*/  wait();  CLS;  } else if (!strcmp(str, "!find")) { /\* Compare the command with "!find" \*/  find(list); /\* Find an athlete in the list \*/  } else if (!strcmp(str, "!sort")) { /\* Compare the command with "!sort" \*/  sort(list); /\* Sort the list of athletes \*/  } else if (!strcmp(str, "!add")) { /\* Compare the command with "!add" \*/  add(list, g\_id++); /\* Add a new athlete to the list \*/  } else if (!strcmp(str, "!edit")) { /\* Compare the command with "!edit" \*/  edit(list); /\* Edit an existing athlete in the list \*/  } else if (!strcmp(str, "!delete")) { /\* Compare the command with "!delete" \*/  delete(list); /\* Delete an athlete from the list \*/  } else if (!strcmp(str, "!save")) { /\* Compare the command with "!save" \*/  save(list); /\* Save the list to a file \*/  } else if (!strcmp(str, "!end")) { /\* Compare the command with "!end" \*/  printf("Goodbye!\n"); /\* Print goodbye message \*/  } else {  printf("Unknown command!\n"); /\* Handle unknown commands \*/  cl = 0; /\* Reset the flag to not display help next time \*/  }  } while (strcmp(str, "!end") != 0); /\* Continue until the user enters "!end" \*/   /\* Free the allocated memory \*/  for (cur\_node = list->first; cur\_node != NULL; cur\_node = cur\_node->next) {  free(cur\_node->data);  free(cur\_node);  }  free(list);  return 0; }  /\* Convert a string to an integer \*/ int from\_str\_to\_int(char \*str) {  int ans, x; /\* Variable to store the resulting integer \*/   ans = 0; /\* Initialize the answer to 0 \*/  while (\*str != '\0' && \*str != '\n') { /\* Loop until the end of the string \*/  x = \*str - '0';  if (0 > x || x > 9) {  ans = 0;  \*str = '\0';  } else {  ans = ans \* 10 + x; /\* Convert character to integer and add to the result \*/  ++str; /\* Move to the next character in the string \*/  }  }  return ans; }  /\* Convert a string to a floating-point number \*/ float from\_str\_to\_float(char \*str) {  float ans, a, b, x; /\* Variables to store the resulting float and decimal places \*/   ans = 0; /\* Initialize the answer to 0 \*/  a = 10; /\* Initialize the factor for integer part \*/  b = 1; /\* Initialize the factor for decimal part \*/  while (\*str != '\0' && \*str != '\n') { /\* Loop until the end of the string \*/  if (\*str == '.' || \*str == ',') { /\* Check for decimal separator \*/  a = 1; /\* Reset the factor for integer part \*/  b = 10; /\* Set the factor for decimal part \*/  ++str; /\* Move to the next character in the string \*/  } else {  x = (float) (\*str - '0');  if (0 > x || x > 9) {  ans = 0;  \*str = '\0';  } else {  ans = ans \* a + x / b; /\* Convert character to float and add to the result \*/  if (b > 1) b \*= 10; /\* Update the decimal factor \*/  ++str; /\* Move to the next character in the string \*/  }  }  }  return ans; }  /\* Convert a string containing delimited integers to an integer array \*/ void from\_str\_to\_int\_mas(char \*str, int \*mas) {  int ind, j; /\* Index variables \*/   ind = 0; /\* Initialize the index for the array \*/  j = 0; /\* Initialize the index for the string \*/  while (str[j] != '\0' && ind < 3) { /\* Loop until the end of the string or the maximum array size \*/  if (str[j] == ';') { /\* Check for delimiter \*/  str[j] = '\0'; /\* Replace delimiter with null terminator \*/  mas[ind++] = from\_str\_to\_int(str); /\* Convert substring to integer and store in the array \*/  str += j + 1; /\* Move to the next substring \*/  j = -1; /\* Reset the index for the substring \*/  }  ++j; /\* Move to the next character in the string \*/  }  if (ind < 3) mas[ind] = from\_str\_to\_int(str); /\* Convert the remaining substring to integer \*/ }  /\* Fill an Athlete structure with data from a string \*/ Athlete \*fill\_struct(char \*str) {  Athlete \*user = NULL; /\* Pointer to the Athlete structure \*/  char \*word, \*pole[5]; /\* Pointers to substrings and an array to store substrings \*/  int ind, tt; /\* Index variables \*/   word = str; /\* Initialize the word pointer to the beginning of the string \*/  ind = 0; /\* Initialize the index for the substring array \*/  user = (Athlete \*) malloc(sizeof(Athlete)); /\* Allocate memory for the Athlete structure \*/  if (user != NULL) { /\* Check if memory allocation is successful \*/  for (tt = 0; str[tt] != '\n' && str[tt] != '\0'; ++tt) { /\* Loop until the end of the string \*/  if (str[tt] == ';' && ind < 5) { /\* Check for delimiter and array bounds \*/  str[tt] = '\0'; /\* Replace delimiter with null terminator \*/  pole[ind++] = word; /\* Store the substring in the array \*/  word = str + tt + 1; /\* Move to the next substring \*/  }  }  if (ind == 5) {  str[tt] = '\0'; /\* Replace the last delimiter with null terminator \*/  user->name = pole[0]; /\* Assign the name to the Athlete structure \*/  user->university = pole[1]; /\* Assign the university to the Athlete structure \*/  user->age = from\_str\_to\_int(pole[2]); /\* Convert and assign the age to the Athlete structure \*/  user->weight = from\_str\_to\_float(pole[3]); /\* Convert and assign the weight to the Athlete structure \*/  user->height = from\_str\_to\_int(pole[4]); /\* Convert and assign the height to the Athlete structure \*/  user->result[0] = 0;  user->result[1] = 0;  user->result[2] = 0;  from\_str\_to\_int\_mas(word, user->result); /\* Convert and assign the result to the Athlete structure \*/  user->index = (float) (user->result[0] + user->result[1] + user->result[2]) / user->weight; /\* Calculate and assign the index to the Athlete structure \*/  } else {  user = NULL;  }  }  return user; }  /\* Create an empty list of athletes \*/ ListOfAthlete \*make\_list() {  ListOfAthlete \*ph = NULL; /\* Pointer to the list \*/   ph = (ListOfAthlete \*) malloc(sizeof(ListOfAthlete)); /\* Allocate memory for the list \*/  if (ph != NULL) { /\* Check if memory allocation is successful \*/  ph->length = 0; /\* Initialize the length of the list to 0 \*/  ph->first = NULL; /\* Initialize the pointer to the first node to NULL \*/  ph->last = NULL; /\* Initialize the pointer to the last node to NULL \*/  }  return ph; }  /\* Create a new node for the list \*/ NodeOfList \*create\_node(char \*str, int g\_id) {  NodeOfList \*new\_node = NULL; /\* Pointer to the new node \*/   new\_node = (NodeOfList \*) malloc(sizeof(NodeOfList)); /\* Allocate memory for the new node \*/  if (new\_node != NULL) { /\* Check if memory allocation is successful \*/  new\_node->data = fill\_struct(strdup(str)); /\* Fill the node with data from the string \*/  new\_node->next = NULL; /\* Initialize the pointer to the next node to NULL \*/  new\_node->prev = NULL; /\* Initialize the pointer to the previous node to NULL \*/  new\_node->id = g\_id; /\* Assign the unique identifier to the node \*/  }  if (new\_node->data == NULL) {  free(new\_node);  new\_node = NULL;  }  return new\_node; }  /\* Display the list of available commands \*/ void help() {  printf("Enter the command:\n"  "\"!print\" = to display the data\n"  "\"!find\" = to find elements of the data\n"  "\"!sort\" = to sort the data\n"  "\"!add\" = to add new data\n"  "\"!edit\" = to edit the data\n"  "\"!delete\" = to remove elements of the data\n"  "\"!save\" = to save the data\n"  "\"!end\" = to end the program\n"); }  /\* Wait for user input to continue \*/ void wait() {  printf("\nTo continue press \"Enter\"...");  getchar(); }  /\* Print a line separator \*/ void print\_line() {  printf("+----+----------------------+------------+-----+--------+--------+------+------+------+-------+\n"); }  /\* Print the header with column names \*/ void print\_head() {  printf("| ID | %-20s | University | Age | Weight | Height | Res1 | Res2 | Res3 | Index |\n", "Name"); }  /\* Print data of a single node \*/ void print\_node(NodeOfList \*node) {  printf("| %-2i | %-20s | %-10s | %-3i | %0.1f ", node->id, node->data->name, node->data->university, node->data->age, node->data->weight);  if (node->data->weight < 100) printf(" ");  if (node->data->weight < 10) printf(" ");  printf("| %-6i | %-4i | %-4i | %-4i | %0.3f |\n", node->data->height, node->data->result[0], node->data->result[1], node->data->result[2], node->data->index); }  /\* Print data of a single node with header and footer \*/ void print\_one(NodeOfList \*node) {  print\_line(); /\* Print line separator \*/  print\_head(); /\* Print header with column names \*/  print\_line(); /\* Print line separator \*/  print\_node(node); /\* Print data of the node \*/  print\_line(); /\* Print line separator \*/ }  /\* Print data of all nodes in the list \*/ void print(ListOfAthlete \*list) {  print\_line(); /\* Print line separator \*/  print\_head(); /\* Print header with column names \*/  print\_line(); /\* Print line separator \*/  for (NodeOfList \*cur\_node = list->first; cur\_node != NULL; cur\_node = cur\_node->next) {  print\_node(cur\_node); /\* Print data of each node \*/  }  print\_line(); /\* Print line separator \*/ }  /\* Sort and print nodes based on a specified parameter \*/ void sorted(int \*mas, ListOfAthlete \*list, int param) {  NodeOfList \*cur\_node, \*min\_node; /\* Pointers to nodes \*/  int ind; /\* Index variable \*/   for (int j = 0; j < list->length; ++j) { /\* Iterate through the list \*/  cur\_node = list->first; /\* Initialize current node \*/  min\_node = NULL; /\* Initialize node with minimum value \*/  for (int i = 0; cur\_node != NULL && i < list->length; ++i, cur\_node = cur\_node->next) { /\* Iterate through the list \*/  if (mas[i] == 1) { /\* Check if the node is not yet sorted \*/  if ((min\_node == NULL) ||  ((param == 1 && min\_node->id > cur\_node->id) || /\* Check for match based on parameter \*/  (param == 2 && strcasecmp(min\_node->data->name, cur\_node->data->name) > 0) ||  (param == 3 && strcasecmp(min\_node->data->university, cur\_node->data->university) > 0) ||  (param == 4 && min\_node->data->age > cur\_node->data->age) ||  (param == 5 && min\_node->data->weight > cur\_node->data->weight) ||  (param == 6 && min\_node->data->height > cur\_node->data->height) ||  (param == 7 && min\_node->data->result[0] > cur\_node->data->result[0]) ||  (param == 8 && min\_node->data->result[1] > cur\_node->data->result[1]) ||  (param == 9 && min\_node->data->result[2] > cur\_node->data->result[2]) ||  (param == 10 && min\_node->data->index > cur\_node->data->index))) {  min\_node = cur\_node; /\* Update node with minimum value \*/  ind = i; /\* Update index of the minimum value \*/  }  }  }  if (min\_node != NULL) { /\* Check if a minimum value node is found \*/  mas[ind] = 2; /\* Mark the node as sorted \*/  print\_node(min\_node); /\* Print the sorted node \*/  }  }  for (int j = 0; j < list->length; ++j) { /\* Iterate through the list \*/  if (mas[j] == 2) mas[j] = 1; /\* Reset sorted nodes \*/  } }  /\* Find and print nodes based on user input \*/ void find(ListOfAthlete \*list) {  NodeOfList \*cur\_node; /\* Pointer to the current node \*/  char str[128]; /\* Buffer for user input \*/  int mas[list->length], fl, param; /\* Array to track sorted nodes, flag, and parameter \*/   CLS; /\* Clear the screen \*/  cur\_node = list->first; /\* Initialize current node \*/  print(list); /\* Print the list \*/  do {  printf("Select a field to find by:\n"  "1 = id\n"  "2 = name\n"  "3 = university\n"  "4 = age\n"  "5 = weight\n"  "6 = height\n"  "7 = result 1\n"  "8 = result 2\n"  "9 = result 3\n"  "10 = index\n"  "0 = exit\n"  "Enter only one number!\n");  fgets(str, sizeof(str), stdin); /\* Read user input \*/  param = from\_str\_to\_int(str); /\* Convert user input to integer \*/  if (param < 0 || 10 < param) { /\* Validate user input \*/  printf("Invalid command!\n");  }  } while (param < 0 || 10 < param); /\* Continue until a valid parameter is selected \*/  if (param != 0) { /\* Check if user wants to exit \*/  printf("Enter the search string:\n"); /\* Prompt for search string \*/  fgets(str, sizeof(str), stdin); /\* Read search string \*/  str[strlen(str) - 1] = '\0'; /\* Remove newline character \*/  CLS; /\* Clear the screen \*/  printf("%s\n", str); /\* Print the search string \*/  strlwr(str); /\* Convert search string to lowercase \*/  fl = 0; /\* Initialize flag \*/  for (int i = 0; cur\_node != NULL && i < list->length; ++i, cur\_node = cur\_node->next) { /\* Iterate through the list \*/  if ((param == 1 && from\_str\_to\_int(str) == cur\_node->id) || /\* Check for match based on parameter \*/  (param == 2 && strstr(strlwr(strdup(cur\_node->data->name)), str) != NULL) ||  (param == 3 && strstr(strlwr(strdup(cur\_node->data->university)), str) != NULL) ||  (param == 4 && from\_str\_to\_int(str) == cur\_node->data->age) ||  (param == 5 && from\_str\_to\_float(str) == cur\_node->data->weight) ||  (param == 6 && from\_str\_to\_int(str) == cur\_node->data->height) ||  (param == 7 && from\_str\_to\_int(str) == cur\_node->data->result[0]) ||  (param == 8 && from\_str\_to\_int(str) == cur\_node->data->result[1]) ||  (param == 9 && from\_str\_to\_int(str) == cur\_node->data->result[2]) ||  (param == 10 && fabsf(from\_str\_to\_float(str) - cur\_node->data->index) < 0.001)) {  if (fl == 0) { /\* Check if matches are found \*/  print\_line(); /\* Print line separator \*/  print\_head(); /\* Print header \*/  print\_line(); /\* Print line separator \*/  }  print\_node(cur\_node); /\* Print the matching node \*/  fl = 1; /\* Set flag to indicate matches found \*/  mas[i] = 1; /\* Mark node as found \*/  } else {  mas[i] = 0; /\* Mark node as not found \*/  }  }  if (fl == 0) { /\* If no matches are found \*/  printf("No matches found!\n"); /\* Print message \*/  wait();  } else {  print\_line(); /\* Print line separator \*/  do {  printf("Select a field to sort by or exit:\n"  "1 = id\n"  "2 = name\n"  "3 = university\n"  "4 = age\n"  "5 = weight\n"  "6 = height\n"  "7 = result 1\n"  "8 = result 2\n"  "9 = result 3\n"  "10 = index\n"  "0 = exit\n"  "Enter only one number!\n");  fgets(str, sizeof(str), stdin); /\* Read user input \*/  param = from\_str\_to\_int(str); /\* Convert input to integer \*/  if (param < 0 || 10 < param) { /\* Validate input \*/  printf("Invalid command!\n");  } else if (param != 0) { /\* If valid sort parameter \*/  print\_line(); /\* Print line separator \*/  print\_head(); /\* Print header \*/  print\_line(); /\* Print line separator \*/  sorted(mas, list, param); /\* Sort and print nodes \*/  print\_line(); /\* Print line separator \*/  }  } while (param != 0); /\* Continue until user exits \*/  }  }  CLS; /\* Clear the screen \*/ }  /\*Creates an array of pointers to NodeOfList structures based on the given list\*/ NodeOfList \*\*get\_mas(ListOfAthlete \*list) {  NodeOfList \*cur\_node; /\* Pointer to traverse the list \*/  NodeOfList \*\*mas = NULL; /\* Array to hold pointers to list nodes \*/   cur\_node = list->first; /\* Start from the first node of the list \*/  mas = (NodeOfList \*\*) malloc(list->length \* sizeof(NodeOfList \*)); /\* Allocate memory for the array of NodeOfList pointers \*/   if (mas != NULL) {  for (int i = 0; cur\_node != NULL; ++i) { /\* Iterate through the list and fill the array with pointers to the nodes \*/  mas[i] = cur\_node; /\* Assign the current node to the array \*/  cur\_node = cur\_node->next; /\* Move to the next node in the list \*/  }  }  return mas; }  /\* Swaps two nodes in the list and updates their positions \*/ void my\_swap(NodeOfList \*\*mas, ListOfAthlete \*list, int i, int j) {  NodeOfList \*q;   if (mas[i] == list->first) { /\* Update the list's first pointer and set the previous pointer of mas[j] to NULL \*/  list->first = mas[j];  mas[j]->prev = NULL;  } else { /\* Update the next pointer of the previous node and set the previous pointer of mas[j] \*/  mas[i - 1]->next = mas[j];  mas[j]->prev = mas[i - 1];  }  if (mas[j] == list->last) { /\* Update the list's last pointer and set the next pointer of mas[i] to NULL \*/  list->last = mas[i];  mas[i]->next = NULL;  } else { /\* Update the previous pointer of the next node and set the next pointer of mas[i] \*/  mas[j + 1]->prev = mas[i];  mas[i]->next = mas[j + 1];  }  if (j - i == 1) { /\* If j and i are adjacent, swap their next and previous pointers \*/  mas[i]->prev = mas[j];  mas[j]->next = mas[i];  } else { /\* Update the next and previous pointers of the nodes surrounding mas[i] and mas[j] \*/  mas[j - 1]->next = mas[i];  mas[i]->prev = mas[j - 1];  mas[i + 1]->prev = mas[j];  mas[j]->next = mas[i + 1];  }  /\* Swap the positions of mas[i] and mas[j] in the array \*/  q = mas[i];  mas[i] = mas[j];  mas[j] = q; }  /\* Sorts the list of athletes based on a selected parameter \*/ void sort(ListOfAthlete \*list) {  NodeOfList \*\*mas; /\* Array to hold pointers to the list nodes \*/  char str[128]; /\* Buffer to read user input \*/  int n, param; /\* n: number of nodes, param: sorting parameter \*/   CLS; /\* Clear the screen \*/  n = list->length; /\* Get the length of the list \*/  mas = get\_mas(list); /\* Create an array of nodes \*/  print(list); /\* Print the list \*/  do { /\* Loop to get the sorting parameter from the user \*/  printf("Select a field to sort by or exit:\n"  "1 = id\n"  "2 = name\n"  "3 = university\n"  "4 = age\n"  "5 = weight\n"  "6 = height\n"  "7 = result 1\n"  "8 = result 2\n"  "9 = result 3\n"  "10 = index\n"  "0 = exit\n"  "Enter only one number!\n");  fgets(str, sizeof(str), stdin); /\* Read the sorting parameter \*/  param = from\_str\_to\_int(str); /\* Convert the input to an integer \*/  if (param < 0 || param > 10) { /\* Check for valid input \*/  printf("Invalid command!\n");  } else if (param != 0) {  for (int i = 0; i < n; ++i) {  for (int j = i + 1; j < n; ++j) {  if ((param == 1 && mas[i]->id > mas[j]->id) || /\* Check if nodes should be swapped based on the selected parameter \*/  (param == 2 && strcasecmp(mas[i]->data->name, mas[j]->data->name) > 0) ||  (param == 3 && strcasecmp(mas[i]->data->university, mas[j]->data->university) > 0) ||  (param == 4 && mas[i]->data->age > mas[j]->data->age) ||  (param == 5 && mas[i]->data->weight > mas[j]->data->weight) ||  (param == 6 && mas[i]->data->height > mas[j]->data->height) ||  (param == 7 && mas[i]->data->result[0] > mas[j]->data->result[0]) ||  (param == 8 && mas[i]->data->result[1] > mas[j]->data->result[1]) ||  (param == 9 && mas[i]->data->result[2] > mas[j]->data->result[2]) ||  (param == 10 && mas[i]->data->index > mas[j]->data->index)) {  my\_swap(mas, list, i, j); /\* Swap the nodes \*/  }  }  }  print(list); /\* Print the sorted list \*/  }  } while (param != 0); /\* Continue until the user exits \*/   free(mas); /\* Free the allocated memory \*/  CLS; /\* Clear the screen \*/ }  /\* Adds a new athlete to the list \*/ void add(ListOfAthlete \*list, int g\_id) {  char str[1024]; /\* Buffer to store user input \*/  NodeOfList \*cur\_node; /\* Pointer to the newly created node \*/   CLS; /\* Clear the screen \*/  print(list); /\* Print the current list of athletes \*/  printf("Enter data of the athlete in the format:\n"  "name;university;age;weight;height;result1;result2;result3\n");  fgets(str, sizeof(str), stdin); /\* Read user input \*/  cur\_node = create\_node(str, g\_id); /\* Create a new node with the given data \*/  if (cur\_node != NULL) { /\* If the new node is created successfully \*/  cur\_node->prev = list->last; /\* Link the previous node to the new node \*/  if (list->length == 0) {  list->first = cur\_node; /\* If the list is empty, set the new node as the first node \*/  } else {  list->last->next = cur\_node; /\* Otherwise, link the last node to the new node \*/  }  list->last = cur\_node; /\* Update the last pointer to the new node \*/  ++list->length; /\* Increment the length of the list \*/  ++g\_id;  printf("The item has been successfully inserted!\n"); /\* Inform the user about the successful insertion \*/  }  print(list); /\* Print the updated list \*/  wait();  CLS; /\* Clear the screen \*/ }  /\* Allows editing the details of an athlete in the list based on the provided ID \*/ void edit(ListOfAthlete \*list) {  NodeOfList \*the\_node = NULL; /\* Pointer to the node to be edited \*/  int id; /\* ID of the athlete to be edited \*/  char str[128], s\_id[128]; /\* Buffers for user input \*/   CLS; /\* Clear the screen \*/  print(list); /\* Print the current list of athletes \*/  printf("Enter the ID of the Athlete you want to edit or 0 to exit:\n");  fgets(s\_id, sizeof(s\_id), stdin); /\* Read the ID input from the user \*/  id = from\_str\_to\_int(s\_id); /\* Convert the input to an integer \*/  for (NodeOfList \*cur\_node = list->first; cur\_node != NULL; cur\_node = cur\_node->next) { /\* Find the node with the specified ID \*/  if (cur\_node->id == id) {  the\_node = cur\_node;  print\_one(the\_node); /\* Print the details of the selected athlete \*/  }  }  if (the\_node != NULL) { /\* If the node with the specified ID is found \*/  /\* Prompt the user to edit each attribute and update if necessary \*/  printf("\nCurrent name: %s\n"  "Write new name or skip (press \"Enter\"):\n", the\_node->data->name);  fgets(str, sizeof(str), stdin);  if (strcmp(str, "\n") != 0) {  str[strcspn(str, "\n")] = '\0';  the\_node->data->name = strdup(str);  }   printf("\nCurrent university: %s\n"  "Write new university or skip (press \"Enter\"):\n", the\_node->data->university);  fgets(str, sizeof(str), stdin);  if (strcmp(str, "\n") != 0) {  str[strcspn(str, "\n")] = '\0';  the\_node->data->university = strdup(str);  }   printf("\nCurrent age: %i\n"  "Write new age or skip (press \"Enter\"):\n", the\_node->data->age);  fgets(str, sizeof(str), stdin);  if (strcmp(str, "\n") != 0) {  the\_node->data->age = from\_str\_to\_int(str);  }   printf("\nCurrent weight: %0.1f\n"  "Write new weight or skip (press \"Enter\"):\n", the\_node->data->weight);  fgets(str, sizeof(str), stdin);  if (strcmp(str, "\n") != 0) {  the\_node->data->weight = from\_str\_to\_float(str);  }   printf("\nCurrent height: %i\n"  "Write new height or skip (press \"Enter\"):\n", the\_node->data->height);  fgets(str, sizeof(str), stdin);  if (strcmp(str, "\n") != 0) {  the\_node->data->height = from\_str\_to\_int(str);  }   printf("\nCurrent Res1: %i\n"  "Write new Res1 or skip (press \"Enter\"):\n", the\_node->data->result[0]);  fgets(str, sizeof(str), stdin);  if (strcmp(str, "\n") != 0) {  the\_node->data->result[0] = from\_str\_to\_int(str);  }   printf("\nCurrent Res2: %i\n"  "Write new Res2 or skip (press \"Enter\"):\n", the\_node->data->result[1]);  fgets(str, sizeof(str), stdin);  if (strcmp(str, "\n") != 0) {  the\_node->data->result[1] = from\_str\_to\_int(str);  }   printf("\nCurrent Res3: %i\n"  "Write new Res3 or skip (press \"Enter\"):\n", the\_node->data->result[2]);  fgets(str, sizeof(str), stdin);  if (strcmp(str, "\n") != 0) {  the\_node->data->result[2] = from\_str\_to\_int(str);  }   the\_node->data->index = (float) (the\_node->data->result[0] + the\_node->data->result[1] + the\_node->data->result[2]) / the\_node->data->weight;  print\_one(the\_node); /\* Print the updated details of the athlete \*/  } else if (id != 0) {  printf("Element not found!\n"); /\* Notify the user if the ID is not found \*/  }  wait();  CLS; /\* Clear the screen \*/ }  /\* Deleting athletes from the list based on the specified parameter \*/ void delete(ListOfAthlete \*list) {  NodeOfList \*cur\_node, \*prev\_node; /\* Pointers to current and previous nodes \*/  char str[128], ch; /\* Buffers for user input \*/  int mas[list->length], fl, param, cnt; /\* Array to mark matched athletes, flag, parameter, count \*/   CLS; /\* Clear the screen \*/  cur\_node = list->first; /\* Start from the first node \*/  print(list); /\* Print the current list of athletes \*/  do { /\* Prompt the user to select a field to delete by \*/  printf("Select a field to delete by:\n"  "1 = id\n"  "2 = name\n"  "3 = university\n"  "4 = age\n"  "5 = weight\n"  "6 = height\n"  "7 = result 1\n"  "8 = result 2\n"  "9 = result 3\n"  "10 = index\n"  "0 = exit\n"  "Enter only one number!\n");  fgets(str, sizeof(str), stdin);  param = from\_str\_to\_int(str); /\* Convert user input to an integer \*/  if (param < 0 || 10 < param) {  printf("Invalid command!\n");  }  } while (param < 0 || 10 < param);  if (param != 0) { /\* If the user chooses to delete \*/  printf("Enter the delete string:\n");  fgets(str, sizeof(str), stdin); /\* Read the delete string from the user \*/  CLS; /\* Clear the screen \*/  printf("%s", str); /\* Print the delete string \*/  str[strcspn(str, "\n")] = '\0'; /\* Remove the newline character \*/  strlwr(str); /\* Convert the delete string to lowercase \*/  fl = 0; /\* Reset the flag \*/  for (int i = 0; cur\_node != NULL && i < list->length; ++i) {  if ((param == 1 && from\_str\_to\_int(str) == cur\_node->id) || /\* Check for match based on parameter \*/  (param == 2 && strstr(strlwr(strdup(cur\_node->data->name)), str) != NULL) ||  (param == 3 && strstr(strlwr(strdup(cur\_node->data->university)), str) != NULL) ||  (param == 4 && from\_str\_to\_int(str) == cur\_node->data->age) ||  (param == 5 && from\_str\_to\_float(str) == cur\_node->data->weight) ||  (param == 6 && from\_str\_to\_int(str) == cur\_node->data->height) ||  (param == 7 && from\_str\_to\_int(str) == cur\_node->data->result[0]) ||  (param == 8 && from\_str\_to\_int(str) == cur\_node->data->result[1]) ||  (param == 9 && from\_str\_to\_int(str) == cur\_node->data->result[2]) ||  (param == 10 && fabsf(from\_str\_to\_float(str) - cur\_node->data->index) < 0.001)) {  if (fl == 0) { /\* Check if matches are found \*/  print\_line(); /\* Print line separator \*/  print\_head(); /\* Print header \*/  print\_line(); /\* Print line separator \*/  }  print\_node(cur\_node); /\* Print the details of the matched athlete \*/  fl = 1; /\* Set the flag \*/  mas[i] = 1; /\* Mark the matched athlete \*/  } else {  mas[i] = 0; /\* Mark as unmatched \*/  }  cur\_node = cur\_node->next; /\* Move to the next node \*/  }  if (fl == 0) { /\* If matches are found \*/  printf("No matches found!\n");  wait();  } else {  print\_line();  printf("Are you sure want to delete these athletes? (Y/N)\n");  do { /\* Prompt for confirmation \*/  ch = (char) getchar();  getchar();  if (ch != 'Y' && ch != 'N') printf("Invalid command!\n");  } while (ch != 'Y' && ch != 'N');  if (ch == 'Y') { /\* If confirmed to delete \*/  cnt = 0; /\* Initialize counter \*/  cur\_node = list->first; /\* Start from the first node \*/  for (int i = 0; i < list->length; ++i) { /\* Iterate through the list and delete the matched athletes \*/  if (mas[i] == 1) {  cnt++; /\* Increment counter \*/  /\* Adjust pointers and free memory \*/  if (cur\_node->prev == NULL) { /\* First node \*/  cur\_node = cur\_node->next;  list->first = cur\_node; /\* Update the first node pointer \*/  free(cur\_node->prev->data); /\* Free memory occupied by data of the deleted node \*/  free(cur\_node->prev); /\* Free memory occupied by the deleted node \*/  cur\_node->prev = NULL; /\* Set the prev pointer of the new first node to NULL \*/  } else if (cur\_node->next == NULL) { /\* Last node \*/  cur\_node = cur\_node->prev; /\* Move to the previous node \*/  list->last = cur\_node; /\* Update the last node pointer \*/  free(cur\_node->next->data); /\* Free memory occupied by data of the deleted node \*/  free(cur\_node->next); /\* Free memory occupied by the deleted node \*/  cur\_node->next = NULL; /\* Set the next pointer of the new last node to NULL \*/  } else { /\* Another node \*/  cur\_node->next->prev = cur\_node->prev; /\* Update the prev pointer of the next node \*/  cur\_node->prev->next = cur\_node->next; /\* Update the next pointer of the previous node \*/  prev\_node = cur\_node; /\* Save a reference to the node to be deleted \*/  cur\_node = cur\_node->next; /\* Move to the next node \*/  free(prev\_node->data); /\* Free memory occupied by data of the deleted node \*/  free(prev\_node); /\* Free memory occupied by the deleted node \*/  }  } else {  cur\_node = cur\_node->next; /\* Move to the next node in the list \*/  }  }  list->length -= cnt; /\* Update the length of the list \*/  print(list); /\* Print the updated list \*/  wait();  }  }  }  CLS; /\* Clear the screen \*/ }  /\* Save the data of athletes stored in a linked list to a file specified by the user \*/ void save(ListOfAthlete \*list) {  FILE \*f; /\* File pointer for saving data \*/  char filename[128]; /\* Array to store the filename entered by the user \*/   CLS; /\* Clear the screen \*/  print(list); /\* Print the list of athletes \*/  printf("Please enter the name of the file to save the data to:\n"); /\* Prompt the user to enter the filename \*/  fgets(filename, sizeof(filename), stdin); /\* Read the filename from the user \*/  filename[strcspn(filename, "\n")] = '\0'; /\* Remove the newline character from the filename \*/  f = fopen(filename, "w"); /\* Open the file for writing \*/  for (NodeOfList \*cur\_node = list->first; cur\_node != NULL; cur\_node = cur\_node->next) {  /\* Write the data of each athlete to the file in the specified format \*/  fprintf(f, "%s;%s;%i;%0.1f;%i;%i;%i;%i\n", cur\_node->data->name, cur\_node->data->university, cur\_node->data->age, cur\_node->data->weight, cur\_node->data->height, cur\_node->data->result[0], cur\_node->data->result[1], cur\_node->data->result[2]);  }  fclose(f); /\* Close the file \*/  printf("The file has been successfully written!\n"); /\* Inform the user that the file has been written successfully \*/  wait();  CLS; /\* Clear the screen \*/ }** |

# Контрольные примеры

|  |  |  |
| --- | --- | --- |
| № | Исходные данные | Результаты |
| 1 |  |  |
| 2 |  |  |
| 3 |  |  |

# Содержимое файлов

# Примеры выполнения программы

# Выводы

Создана электронная картотека и программа на языке Си, обеспечивающая взаимодействие с ней. Реализованы все указанные в задании функции. Создано понятное пользователю меню. Программа работоспособна и протестирована на Windows и Linux.

В работе использованы следующие заголовочные файлы стандартной библиотеки:

* *<stdio.h>* - используется для ввода и вывода из файла и консоли.
* *<stdlib.h>* - используется для выделения памяти для списка и динамических массивов.
* *<string.h>* - используется для обработки и преобразования строк.
* *<math.h>* - используется для поиска модуля числа.