МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра систем автоматизированного проектирования

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

по дисциплине «Программирование»

Студент гр. 3353 _____ Карпенко А.Ю.

Преподаватель _____ Калмычков В.А.

Санкт-Петербург

2024

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

по дисциплине «Программирование»

Студент: Карпенко А.Ю.
Группа: 3353
Тема работы: Информационная система

Исходные данные: Менеджер футбольной команды имеет данные об играющих за клуб игроках и об игроках, изъявивших желание играть в команде. Надо хранить информацию: ФИО, дата рождения, адрес, амплуа, статус (член команды или кандидат), послужной список (команды, в которых играл, число игр, забитых и/или пропущенных голов и голевых передач за каждую из них, для играющих дата оформления в команду). Необходимо получить списки: игроков, сгруппированные по позиции и возрастной категории; игроков и кандидатов, которые играли ранее в одних командах; учетные карточки на каждого игрока, упорядоченные (отдельно) по общему числу игр, голов и голевых передач за все команды; учетные карточки на каждого игрока команды, упорядоченные (отдельно) по общему числу игр, голов и голевых передач за команду.

Содержание пояснительной записки: аннотация, анализ задания, выполнение контрольного примера, математическая постановка задачи, особенности выполнения на компьютере, представление алгоритма решения задачи, текст программы, тесты программы

Предполагаемый объем пояснительной записки:

Не менее 20 страниц.

Дата выдачи задания: 20.02.2024

Дата сдачи реферата: ___ _____2024

Дата защиты реферата: ___ _____2024

Студент Карпенко А.Ю.

Преподаватель Калмычков В.А

Аннотация

Результатом выполнения курсового проекта стала программа обеспечивающая хранение информации игроков футбольной команды, обеспечивающая вывод списков, отсортированных по заданным критериям. Выбор нужного запроса и критерия осуществляется через меню в консоли.

Тем самым курсовая работа стала сборной реализаций практических навыков, полученных за семестр, тем самым закрепив их и продемонстрировав владение ими.

Курсовой проект состоит из программы на C++ и пояснительной записки. Пояснительная записка включает в себя разделы - исходная формулировка, цель, математическая формулировка, контрольный пример, организация ввода-вывода, организация хранения данных, представление алгоритма решения, текст программы, тесты программы.

Annotation

The result of the course project was a program that stores information from the players of the football team, providing lists sorted by given criteria. The selection of the desired query and criterion is carried out through the menu in the console.

Thus, coursework became a team of implementations of practical skills gained per semester, thereby securing them and demonstrating their mastery.

The course project consists of a C++ program and an explanatory note. The explanatory note includes sections - initial formulation, purpose, mathematical formulation, test example, I/O organization, data storage organization, solution algorithm presentation, program text, program tests.

Оглавление

Исходная формулировка задания
Менеджер футбольной команды имеет данные об играющих за клуб игроках и об игроках, изъявивших желание играть в команде. Надо хранить информацию: ФИО, дата рождения, адрес, амплуа, статус (член команды или кандидат), послужной список (команды, в которых играл, число игр, забитых и/или пропущенных голов и голевых передач за каждую из них, для играющих дата оформления в команду). Необходимо получить списки: игроков, сгруппированные по позиции и возрастной категории; игроков и кандидатов, которые играли ранее в одних командах; учетные карточки на каждого игрока, упорядоченные (отдельно) по общему числу игр, голов и голевых передач за все команды; учетные карточки на каждого игрока команды, упорядоченные (отдельно) по общему числу игр, голов и голевых передач за команду.

Цель работы
Продемонстрировать навыки и знания, приобретенные в течении семестра в том числе: умение работать с списками, классами и разделением программы на файлы, а также различными представлениями данных

Контрольный пример
В качестве контрольного примера составим список игроков со всеми необходимыми данными и составим требующиеся списки. Все операции заполнения, сортировки должны сработать корректно.

Разработка интерфейса пользователя
При запуске программы в консоли появляется главное меню, позволяющее выбрать нужный список
Select a function:
1. list of players grouped by position and age category
2. list of players and candidates who have previously played in the same teams
3. scorecards for each player, ordered (separately) by the total number of games, goals and assists for all teams
4. registration cards for each player of the team, ordered (separately) by the total number of games, goals and assists for the team
5. exit
Your choice:
При выборе 3 варианта появляется подменю. Необходимо выбрать критерий, по которому будет проходить сортировка.
1. comparison of players by total number of matches
2. comparison of players by total number of goals scored
3. comparison of players by the total number of goals conceded
4. comparison of players by total number of assists
5. [Back to main menu]
Enter your choice:
В подменю выбор 5 означает возвращение к главному меню программы. Если ввести любую другую цифру, кроме 1-5, программа выдаст информацию об ошибочном вводе и предоставит возможность повторить выбор.
Аналогично работает и главное меню. При любой цифре, кроме 1-5, дается повторный выбор. При 5 программа завершает работу.

Организация ввода-вывода
Программе дается 2 входных файла:

Первый содержит личную информацию об игроке. Данные записаны через «, », что является маркером для разделения полей:
nn, ssss ssss ssss, nnnn-nn-nn, ssss, ssss, ssss
nn, ssss ssss ssss, nnnn-nn-nn, ssss, ssss, ssss
…
nn, ssss ssss ssss, nnnn-nn-nn, ssss, ssss, ssss

Второй файл содержит информацию о командах и играх игрока. Данные также записаны через «, », что является маркером для разделения полей:

nn, ssss, nn, nn, nn, nn
nn, ssss, nn, nn, nn, nn
…
nn, ssss, nn, nn, nn, nn

Программа имеет несколько выходных файлов. На каждый запрос создается свой выходной файл.

- Список игроков, сгруппированный по позиции и возрастной категории;

```
Position      Year    Player
-----------------------------------------------
sssss         nnnn    ssss ssss ssss
              nnnn    ssss ssss ssss
              nnnn    ssss ssss ssss
```

- Список игроков и кандидатов, которые играли ранее в одних командах;

```
Team: ssss
ssss ssss ssss     ssss
ssss ssss ssss     ssss
…
ssss ssss ssss     ssss
-------------------
```

- Учетные карточки на каждого игрока, упорядоченные (отдельно) по общему числу игр, голов и голевых передач за все команды;

```
Player                  Matches      Goals Scored   Goals Conceded Assists
-------------------------------------------------------------------------------------------
ssss ssss ssss          nn           nn             nn             nn
ssss ssss ssss          nn           nn             nn             nn
…
ssss ssss ssss          nn           nn             nn             nn
```

- Учетные карточки на каждого игрока команды, упорядоченные (отдельно) по общему числу игр, голов и голевых передач за команду

```
Player Cards for Team: ssss
Player                  Matches      GoalsScored    GoalsConceded  Assists
ssss ssss ssss          nn           nn             nn             nn
ssss ssss ssss          nn           nn             nn             nn
…
ssss ssss ssss          nn           nn             nn             nn
```

выходные файла лога.

- Файл лога действий

```
(nn) ssss ssss ssss
(nn) ssss ssss ssss
…
(nn) ssss ssss ssss
работа программы завершена
```

- Файл лога памяти

```
Head NULL
---------------------------------------
 | /|\
\|/ |
---------------------------------------
id:       1
name:     Head —> ssss -> ssss -> ssss -> ssss -> ssss -> NULL
year:     nnnn
city:     Head —> ssss -> ssss -> NULL
position:         Head —> ssss -> ssss -> NULL
status:   Head —> ssss -> ssss -> ssss -> NULL
```

teamName: Head —> ssss -> ssss -> NULL
commonPlayedMatches: nn
commonGoalsScored: nn
commonGoalsConceded: nn
commonAssists: nn

Особенности выполнения на компьютере

Тип float представляет вещественное число с плавающей точкой в диапазоне от -3.4E-38 до 3.4E38 и в памяти занимает 4 байта (32 бита), что накладывает свои ограничения на величину переменные этого типа и на элементы массивов в частности. Тип переменной int, который хранит целочисленные значения в диапазоне от –32768 до 32767 и занимает 2 байта (16 бит) накладывает свои огорчения на переменные этого типа. В программе используются как динамические массивы. В файле используются переменке типы char, что накладывает ограничение на символы.

Ключевые слова struct позволяет создать пользовательский тип данных в языке C++, который может содержать как переменные-члены, так и методы. Они включаются в себя свойства(переменные) и методы(функции). В классе все члены по умолчанию public.

Динамический одномерный массив**:**

Объявление:

```
char *A;
```

Выделение памяти:

```
A = new char[n];
```

Обращение к элементу массива:

```
A[номер элемента]
```

Освобождение памяти:

```
delete[] A;
```

В процессе выполнения программы потребуется реализация условных конструкций:

```
if (условие 1) {(выполняется если условие 1 истина);}
else if(условие 2) {выполняется если условие 2 истина;}
else {выполняется если условие 1 И условие 2 ложь;};
```

А также потребуется циклы, будем использовать циклы:

```
while(условие провидения циела)
{
(тело цикла)
}
for(начальные условия; условия продолжения; действие после завершения одного тела)
{
(тело цикла)
```

Объявления конструкции struct:

```
struct *названия конструкции*{
//перемененные
*названия конструкции*(): //конструктор
{
}
~*названия конструкции*(): //деструктор
{
}
// методы
}:
```

Программа дробится на файлы посредством вынесения функций и структур в отдельный файл, таким образом появляются два файла файл расширения (.h) содержащий только сами методы и их аргументы и файл реализации (.cpp) содержащий непосредствен код реализации для методов, в файле расширения обязательно выполнить проверку на повторное включение:

```
#ifndef *название*
#define *название*

*основной код*
```

#endif

Подключение происходит схожим образом, как подключение библиотек:

#include "*название*"

Организация хранения данных

```
struct Player {
    // Конструктор
    Player(int id, char **playerName, int playerYear, char **playerCity, char **playerPosition, char **playerStatus);
    int idPlayer;                       // Идентификатор игрока
    char **name;                        // Имя игрока
    int year;                           // Год рождения игрока
    char **city;                        // Город, в котором игрок родился
    char **position;                    // Позиция игрока на поле
    char **status;                      // Статус игрока
    TeamStatList *statList;                     // Список статистики команды
    int commonPlayedMatches = 0;                // Общее количество сыгранных матчей
    int commonGoalsScored = 0;                  // Общее количество забитых голов
    int commonGoalsConceded = 0;                // Общее количество пропущенных голов
    int commonAssists = 0;                      // Общее количество голевых передач
    ~Player();  // Деструктор
    // Оператор вывода в поток
    friend std::ostream &operator<<(std::ostream &os, const Player &player);
    // Оператор сравнения на равенство
    friend bool operator==(const Player &lhs, const Player &rhs);
};
```

Имя функции	Назначение	параметры		Возвращаемое значение	Внешние эффекты
		входные	выходные		
appendTeamStat	Метод для добавления статистики команды и обновления общих статистических данных игрока	const TeamStat& value	-	void	-
std::ostream &operator<<	Оператор вывода в поток	std::ostream &os, const Player &player	-	-	Вывод данных в поток
operator==	Оператор сравнения на равенство	const Player &lhs, const Player &rhs	-	bool	-

```
struct TeamStat {
    char **teamName;
    int playedMatches = 0;      // Сыгранные матчи
    int goalsScored = 0;        // Забитые голы
    int goalsConceded = 0;      // Пропущенные голы
    int assists = 0;            // Голевые передачи
    // Конструктор
    TeamStat(char **teamName, int matches, int scored, int conceded, int assist);
    // Деструктор
    ~TeamStat();
```

Имя функции	Назначение	параметры		Возвращаемое значение	Внешние эффекты
		входные	выходные		
std::ostream &operator<<	Оператор вывода в поток	std::ostream &os, const Player &player	-	-	Вывод данных в поток
operator==	Оператор сравнения на равенство	const Player &lhs, const Player &rhs	-	bool	-

```
struct IntList {
    IntNode *head; // Указатель на начало списка
    IntNode *tail; // Указатель на конец списка
    // Конструктор по умолчанию
    IntList();
    // Деструктор
    ~IntList();
```

Имя функции	Назначение	параметры		Возвращаемое значение	Внешние эффекты
		входные	выходные		
append	Добавление элемента в список	int value	-	void	-
std::ostream &operator<<	Оператор вывода в поток	std::ostream& os, const IntList& list	-	-	Вывод данных в поток

```
// Класс для списка char*
struct StringList {
    StringNode* head; // Указатель на начало списка
    StringNode* tail; // Указатель на конец списка
    // Конструктор по умолчанию
    StringList();
    // Деструктор
    ~StringList();
```

Имя функции	Назначение	параметры		Возвращаемое значение	Внешние эффекты
		входные	выходные		
StringNode *StringList::append	Добавление элемента в список	const char *value	-	StringNode	-
std::ostream &operator<<	Оператор вывода в поток	std::ostream& os, const StringList& list	-	-	Вывод данных в поток

```
struct PlayerList {
    PlayerNode *head; // Указатель на начало списка
    PlayerNode *tail; // Указатель на конец списка
    // Конструктор по умолчанию
    PlayerList();
    // Деструктор для освобождения памяти
    ~PlayerList();
```

Имя функции	Назначение	параметры		Возвращаемое значение	Внешние эффекты
		входные	выходные		
appendNode	Добавление узла	const Player &value	-	void	-
findById	Сопоставление игроков по id	int playerId	&(currentNode->data)	Player	-
printPlayer	Вывод данных об игроке	const PlayerList &list	-	void	Вывод данных игрока в поток
std::ostream &operator<<	Оператор вывода в поток	std::ostream& os, const PlayerListt& list	-	-	Вывод данных в поток

```
struct TeamStatList {
    TeamStatNode *head; // Указатель на начало списка
    TeamStatNode *tail; // Указатель на конец списка
    // Конструктор по умолчанию
    TeamStatList();
    // Деструктор для освобождения памяти
    ~TeamStatList();
```

Имя функции	Назначение	параметры		Возвращаемое значение	Внешние эффекты
		входные	выходные		
appendNode	Добавление узла	const TeamStat &value	-	void	-
std::ostream &operator<<	Оператор вывода в поток	std::ostream& os, const PlayerListt& list	-	-	Вывод данных в поток

```
struct IntNode {
    int data; // Данные узла
    IntNode* next; // Указатель на следующий узел
    // Конструктор
    explicit IntNode(int value);
    // Деструктор
    ~IntNode();
};

struct StringNode {
    char* data; // Указатель на данные
    StringNode* next; // Указатель на следующий узел
    // Конструктор
    explicit StringNode(const char* value);
    ~StringNode();
};
```

Имя функции	Назначение	параметры		Возвращаемое значение	Внешние эффекты
		входные	выходные		
operator==	Оператор равенства	const StringNode &lhs, const StringNode &rhs	-	bool	-

```
struct PlayerNode {
    Player data; // Исправлено на Position
    PlayerNode *next; // Указатель на следующий узел
    // Конструктор для инициализации узла с заданным значением и указателем на следующий узел
    explicit PlayerNode(const Player& value);
    // Деструктор для освобождения ресурсов
    ~PlayerNode();
```

Имя функции	Назначение	параметры		Возвращаемое значение	Внешние эффекты
		входные	выходные		
operator==	Оператор равенства	const StringNode &lhs, const StringNode &rhs	-	bool	-
std::ostream &operator<<	Оператор вывода в поток	std::ostream& os, const PlayerListt& list	-	-	Вывод данных в поток

```
struct TeamStatNode {
    TeamStat data;
    TeamStatNode *next;
    // Конструктор для инициализации узла с заданным значением и указателем на следующий узел
    explicit TeamStatNode(const TeamStat& value);
    // Деструктор для освобождения ресурсов
    ~TeamStatNode();
};
```

Имя функции	Назначение	параметры		Возвращаемое значение	Внешние эффекты
		входные	выходные		
operator==	Оператор равенства	const TeamStatNode &lhs, const TeamStatNode &rhs	-	bool	-
std::ostream &operator<<	Оператор вывода в поток	std::ostream& os, const TeamStatNode & list	-	-	Вывод данных в поток

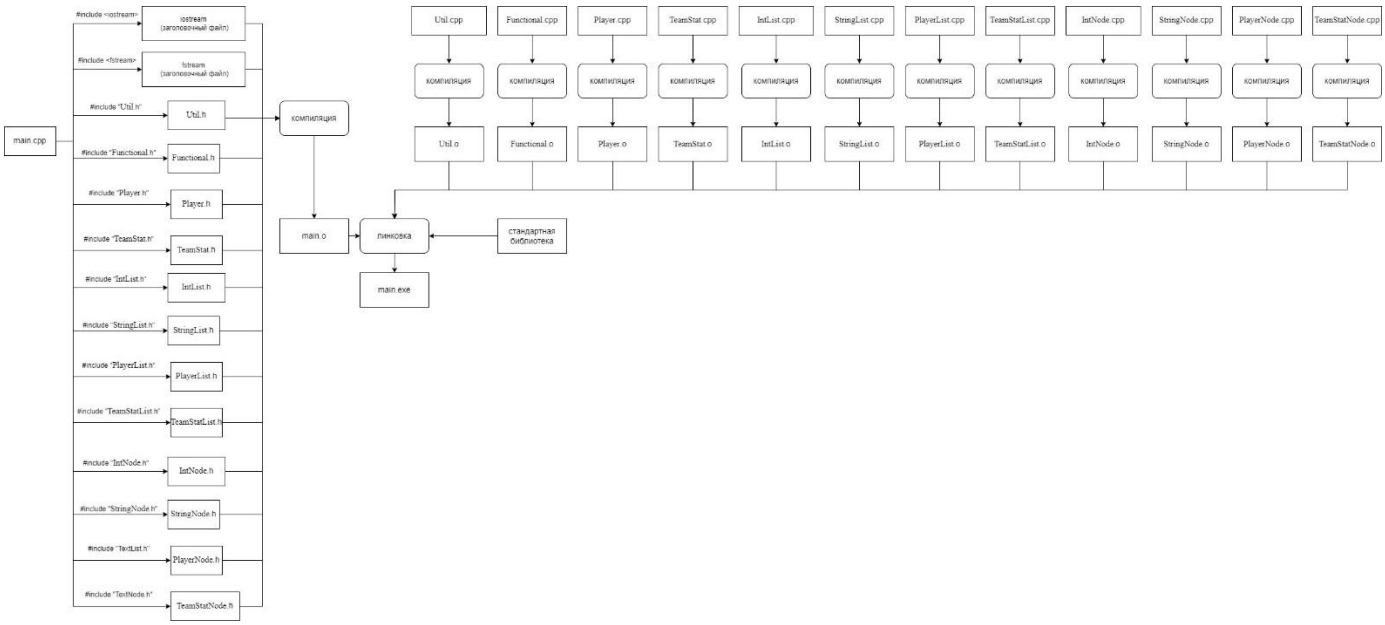
```
//1 игроков, сгруппированные по позиции и возрастной категории
void printPlayersGroupedByPositionAndAgeCategory(StringList *positionList, IntList *dateOfBirthList, PlayerList *playerList);
//2 список игроков и кандидатов, которые играли ранее в одних командах
void printPlayersInTeams(PlayerList *playerList, StringList *teamNameList);
//3 учетные карточки на каждого игрока, упорядоченные (отдельно) по общему числу игр, голов и голевых передач за все команды,
void printPlayerCards(PlayerList *playerList, int choice);
//4 учетные карточки на каждого игрока команды, упорядоченные (отдельно) по общему числу игр, голов и голевых передач за команду
void printPlayerCardsByTeam_1(PlayerList *playerList, StringList *teamNameList);
void printPlayerCardsByTeam_2(PlayerList *playerList, StringList *teamNameList);
void printPlayerCardsByTeam_3(PlayerList *playerList, StringList *teamNameList);
```

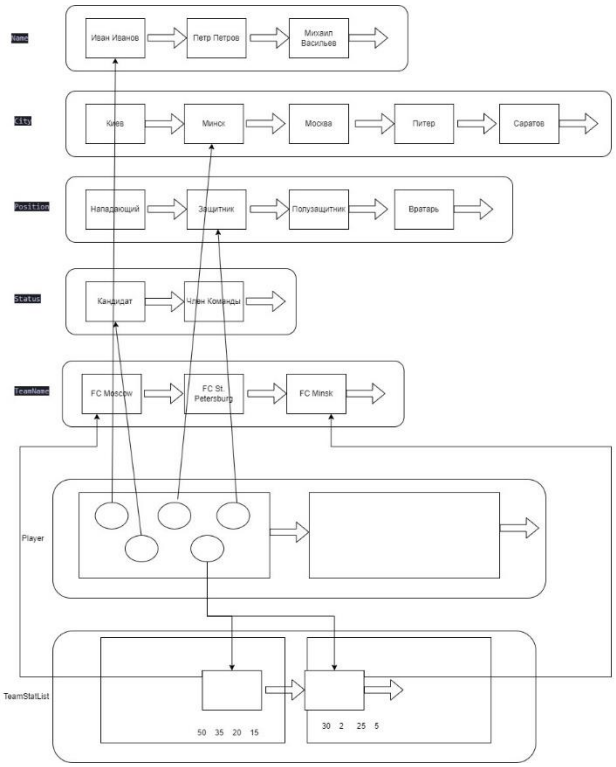
void printPlayerCardsByTeam_4(PlayerList *playerList, StringList *teamNameList);

Имя функции	Назначение	параметры		Возвращаемое значение	Внешние эффекты
		входные	выходные		
printPlayersGroupedByPositionAndAgeCategory	Обработка 1 запроса	StringList *positionList, IntList *dateOfBirthList, PlayerList *playerList	-	Void	Вывод в поток
playedInTeam	Проверка на принадлежность игрока команде	const Player &player, const char *teamName	-	Void	-
printPlayersInTeams	Обработка 2 запроса	PlayerList *playerList, StringList *teamNameList	-	Void	Вывод в поток
compareByPlayedMatches	Функция для сравнения игроков по общему числу игр	const Player &player1, const Player &player2	-	bool	-
swapPlayerNode	Обмен данных игроков	PlayerNode *playerNode, PlayerNode *playerNode2	-	Void	-
sortPlayers	Сортировка для 3 запроса	PlayerList *playerList, bool (*compare)(const Player &player1, const Player &player2)	-	Void	-
printPlayerCards	Обработка 3 запроса	PlayerList *playerList, int choice	-	Void	Вывод в поток
compareBy_PlayedMatches	Функция для сравнения игроков по числу игр	const Player &player1, const Player &player2, const char *teamName	-	bool	-
sortPlayersByTeamName	Сортировка для 4 запроса	PlayerList *playerList, const char *teamName, bool (*compare)(const Player &player1, const Player &player2, const char *teamName)	-	void	-
printPlayerStats	Вывод статистики игрока	std::ofstream &outputFile, PlayerNode *current, const char *teamName, int nameWidth, int size	-	void	Вывод в поток
printPlayerCardsByTeam_1	Обработка 4 запроса	PlayerList *playerList, StringList *teamNameList	-	void	Вывод в поток

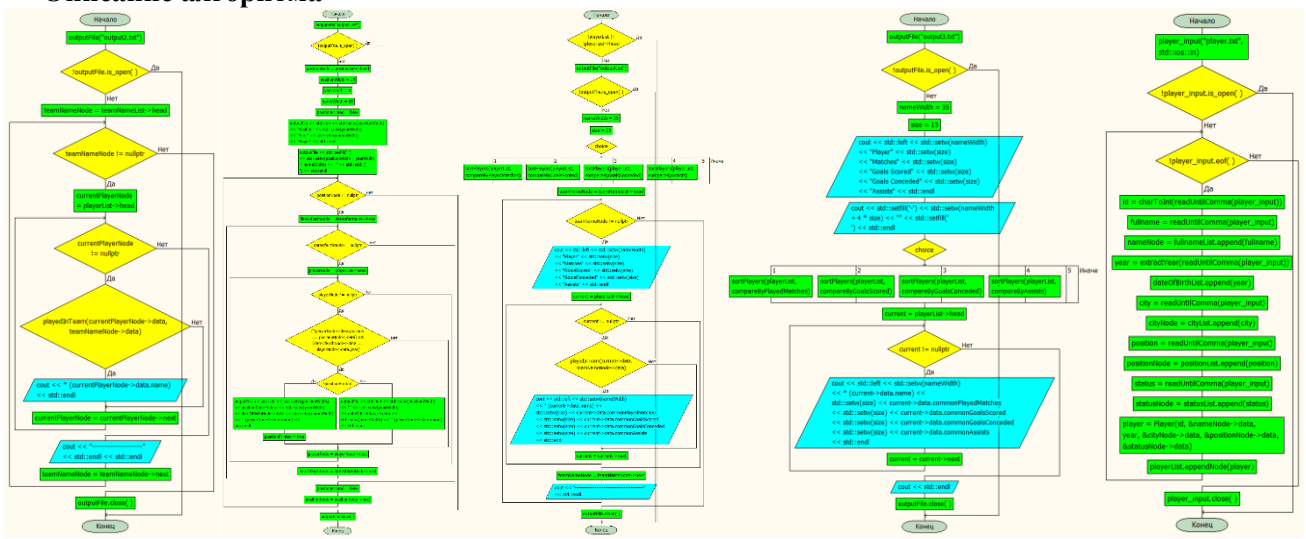
void readPlayersFromFile(PlayerList &playerList, StringList &fullnameList, IntList &dateOfBirthList, StringList &cityList, StringList &positionList, StringList &statusList)
void readTeamsFromFile(PlayerList &playerList, StringList &teamNameList)
void menu(PlayerList &playerList, StringList &positionList, StringList &teamNameList, IntList &dateOfBirthList, std::ofstream &output_activity)

Имя функции	Назначение	параметры		Возвращаемое значение	Внешние эффекты
		входные	выходные		
readPlayersFromFile	Чтение данных игрока из файла	PlayerList &playerList, StringList &fullnameList, IntList &dateOfBirthList, StringList &cityList, StringList &positionList, StringList &statusList	-	void	Открытие файла
readTeamsFromFile	Чтение данных о командах игрока	PlayerList &playerList, StringList &teamNameList	-	void	Открытие файла
menu	меню	PlayerList &playerList, StringList &positionList, StringList &teamNameList, IntList &dateOfBirthList, std::ofstream &output_activity	-	void	Вывод в консоль и файл


#include <iostream>
iostream (заголовочный файл)
#include <fstream>
fstream (заголовочный файл)
#include "Util.h"
Util.h
#include "Functional.h"
Functional.h
#include "Player.h"
Player.h
#include "TeamStat.h"
TeamStat.h
#include "IntList.h"
IntList.h
#include "StringList.h"
StringList.h
#include "PlayerList.h"
PlayerList.h
#include "TeamStatList.h"
TeamStatList.h
#include "IntNode.h"
IntNode.h
#include "StringNode.h"
StringNode.h
#include "TextList.h"
PlayerNode.h
#include "TextNode.h"
TeamStatNode.h
main.cpp
компиляция
main.o
линковка
стандартная библиотека
main.exe
Util.cpp
Functional.cpp
Player.cpp
TeamStat.cpp
IntList.cpp
StringList.cpp
PlayerList.cpp
TeamStatList.cpp
IntNode.cpp
StringNode.cpp
PlayerNode.cpp
TeamStatNode.cpp
компиляция
Util.o
Functional.o
Player.o
TeamStat.o
IntList.o
StringList.o
PlayerList.o
TeamStatList.o
IntNode.o
StringNode.o
PlayerNode.o
TeamStatNode.o


Name
Иван Иванов
Петр Петров
Михаил Васильев
City
Киев
Минск
Москва
Питер
Саратов
Position
Нападающий
Защитник
Полузащитник
Вратарь
Status
Кандидат
Член Команды
TeamName
FC Moscow
FC St Petersburg
FC Minsk
Player
TeamStatList
50 35 20 15
30 2 25 5

Описание алгоритма



Текст программы

main.cpp

```
#include <iostream>
#include <fstream>

#include "model/core/Player.h"

#include "model/list/StringList.h"
#include "model/list/IntList.h"
#include "model/list/PlayerList.h"
#include "Functional.h"

using namespace std;

void
readPlayersFromFile(PlayerList &playerList, StringList &fullnameList, IntList &dateOfBirthList, StringList &cityList,
                    StringList &positionList, StringList &statusList) {
    ifstream player_input("player.txt", std::ios::in);
    if (!player_input.is_open()) {
        std::cerr << "Failed to open the file." << std::endl;
        return;
    }

    while (!player_input.eof()) {
        int id = charToInt(readUntilComma(player_input));

        char *fullname = readUntilComma(player_input);
        auto nameNode = fullnameList.append(fullname);

        int year = extractYear(readUntilComma(player_input));
        dateOfBirthList.append(year);

        char *city = readUntilComma(player_input);
        auto cityNode = cityList.append(city);

        char *position = readUntilComma(player_input);
        auto positionNode = positionList.append(position);

        char *status = readUntilComma(player_input);
        auto statusNode = statusList.append(status);

        Player player = Player(id, &nameNode->data, year, &cityNode->data, &positionNode->data, &statusNode->data);
        playerList.appendNode(player);
    }
    player_input.close();

}

void readTeamsFromFile(PlayerList &playerList, StringList &teamNameList) {

    std::ifstream team_input("team.txt");
    if (!team_input.is_open()) {
        std::cerr << "Failed to open the file." << std::endl;
        return;
    }
    while (!team_input.eof()) {
        int playerId = charToInt(readUntilComma(team_input));
        char *teamName = readUntilComma(team_input);
        int playedMatches = charToInt(readUntilComma(team_input));    // Сыгранные матчи
        int goalsScored = charToInt(readUntilComma(team_input));      // Забитые голы
        int goalsConceded = charToInt(readUntilComma(team_input));    // Пропущенные голы
        int assists = charToInt(readUntilComma(team_input));          // Голевые передачи

        auto teamNameNode = teamNameList.append(teamName);
        //cout << teamName << endl;

        TeamStat teamStat(&teamNameNode->data, playedMatches, goalsScored, goalsConceded, assists);
        //cout << teamStat << endl;
        Player *player = playerList.findById(playerId);
        if (player != nullptr) {
            player->appendTeamStat(teamStat);
        }

    }
}

void handleChoice(int choice3, std::ostream &output_activity) {
    switch (choice3) {
        case 1:
            output_activity
                    << "(3.1) сформированы учетные карточки на каждого игрока, упорядоченные по общему числу игр за все команды"
                    << std::endl;
            break;
        case 2:
            output_activity
                    << "(3.2) сформированы учетные карточки на каждого игрока, упорядоченные по общему числу забитых голов за все команды"
                    << std::endl;
            break;
        case 3:
            output_activity
                    << "(3.3) сформированы учетные карточки на каждого игрока, упорядоченные по общему числу пропущенных голов за все команды"
                    << std::endl;
            break;
        case 4:
            output_activity
                    << "(3.4) сформированы учетные карточки на каждого игрока, упорядоченные по общему числу голевых передач за все команды"
                    << std::endl;
            break;
        case 5:
            output_activity
                    << "(3.5) возвращение к общему меню"
                    << std::endl;
            break;
        default:
            output_activity
                    << "(3) Неправильный выбор. Попробуйте снова"
                    << std::endl;
            break;
    }
}

void menu(PlayerList &playerList, StringList &positionList, StringList &teamNameList, IntList &dateOfBirthList,
          std::ofstream &output_activity) {
    int choice;
    bool exitMenu = false;

    while (!exitMenu) {
        // Выводим меню
        std::cout << "Select a function:" << std::endl;
        //1 игроков, сгруппированные по позиции и возрастной категории
        std::cout << "1. list of players grouped by position and age category" << std::endl;
        //2 список игроков и кандидатов, которые играли ранее в одних командах
        std::cout << "2. list of players and candidates who have previously played in the same teams" << std::endl;
        //3 учетные карточки на каждого игрока, упорядоченные (отдельно) по общему числу игр, голов и голевых передач за все команды,
        std::cout
                << "3. scorecards for each player, ordered (separately) by the total number of games, goals and assists for all teams"
                << std::endl;
        //4 учетные карточки на каждого игрока команды, упорядоченные (отдельно) по общему числу игр, голов и голевых передач за команду
        std::cout
                << "4. registration cards for each player of the team, ordered (separately) by the total number of games, goals and assists for the team"
                << std::endl;
        std::cout << "5. exit" << std::endl;
        std::cout << "Your choice: ";
        std::cin >> choice;

        // Обрабатываем выбор пользователя
        switch (choice) {
            case 1:
                printPlayersGroupedByPositionAndAgeCategory(&positionList, &dateOfBirthList, &playerList);
                output_activity << "(1) сформирован список игроков, сгруппированный по позиции и возрастной категории"
                                << std::endl;
                break;
            case 2:
                printPlayersInTeams(&playerList, &teamNameList);
                output_activity << "(2) сформирован список игроков и кандидатов, которые играли ранее в одних командах"
                                << std::endl;
                break;
            case 3:
                int choice3;
                do {
                    // Отобразить опции для выбора
```

```
                std::cout << "1. comparison of players by total number of matches" << std::endl;
                std::cout << "2. comparison of players by total number of goals scored" << std::endl;
                std::cout << "3. comparison of players by the total number of goals conceded" << std::endl;
                std::cout << "4. comparison of players by total number of assists" << std::endl;

                std::cout << "5. [Back to main menu]" << std::endl;
                std::cout << "Enter your choice: ";
                std::cin >> choice3;
                printPlayerCards(&playerList, choice3);
                handleChoice(choice3, output_activity);
            } while (choice3 != 5); // Повторять до тех пор, пока пользователь не выберет "Вернуться в главное меню"
            break;
        case 4:
            printPlayerCardsByTeam_1(&playerList, &teamNameList);
            printPlayerCardsByTeam_2(&playerList, &teamNameList);
            printPlayerCardsByTeam_3(&playerList, &teamNameList);
            printPlayerCardsByTeam_4(&playerList, &teamNameList);
            output_activity
                << "(4) сформированы учетные карточки на каждого игрока команды, упорядоченные по общему числу игр, забитых и пропущенных голов, голевых передач за команду";
            output_activity << std::endl;
            break;
        case 5:
            exitMenu = true;
            output_activity << "работа программы завершена" << std::endl;
            break;
        default:
            std::cout << "Wrong choice. Try again" << std::endl;
            output_activity << "Неправильный выбор. Попробуйте снова" << std::endl;
            break;
    }
  }
}


int main() {
  StringList fullnameList;
  IntList dateOfBirthList;
  StringList cityList;
  StringList positionList;
  StringList statusList;

  PlayerList playerList;

  StringList teamNameList;

  std::ofstream output_activity("output_activity.txt");
  if (!output_activity.is_open()) {
    std::cerr << "Error: Unable to open output file." << std::endl;
    return 1;
  }
  readPlayersFromFile(playerList, fullnameList, dateOfBirthList, cityList, positionList, statusList);

  readTeamsFromFile(playerList, teamNameList);

  PlayerList::printPlayer(playerList);
  menu(playerList, positionList, teamNameList, dateOfBirthList, output_activity);

  return 0;
}
```

Util.h

```
#ifndef PROGA_KURS_UTIL_H
#define PROGA_KURS_UTIL_H

#include <iostream>
#include <fstream>

int length(const char *str);

void copy(const char *source, char *destination, int len);

int extractYear(const char *dateStr);

int strcmp(const char *str1, const char *str2);

char *readUntilComma(std::ifstream &input);

int charToInt(const char *str);
void printString(std::ofstream &output, const char *str);
#endif //PROGA_KURS_UTIL_H
```

Util.cpp

```
#include <iostream>
#include <fstream>

// Функция для определения длины строки
int length(const char *str) {
  int len = 0;
  while (str[len] != '\0') {
    len++;
  }
  return len;
}

// Функция для копирования строки
void copy(const char *source, char *destination, int len) {
  for (int i = 0; i < len; ++i) {
    destination[i] = source[i];
  }
  destination[len] = '\0'; // Добавляем завершающий нулевой символ
}

int extractYear(const char *dateStr) {
  int year = 0;
  // Ищем позицию первого символа '-'
  int pos = 0;
  while (dateStr[pos] != '-' && dateStr[pos] != '\0') {
    pos++;
  }
  // Переводим символы до первого '-' в число
  for (int i = 0; i < pos; ++i) {
    if (dateStr[i] >= '0' && dateStr[i] <= '9') {
      year = year * 10 + (dateStr[i] - '0');
    }
  }
  return year;
}

int strcmp(const char *str1, const char *str2) {
  while (*str1 && *str2 && *str1 == *str2) {
    ++str1;
    ++str2;
  }
  return *str1 - *str2;
}

char *readUntilComma(std::ifstream &input) {
  const int bufferSize = 100; // Максимальная длина строки
  char *buffer = new char[bufferSize];
  int index = 0;
  char symbol;

  while (input.get(symbol) && symbol != EOF && symbol != ',' && symbol != '\n') {
    buffer[index++] = symbol;
    if (index >= bufferSize - 1) // Проверка на переполнение буфера
      break;
  }
  if (symbol == ',') {
    input.get(symbol);
  }
  buffer[index] = '\0'; // Добавляем завершающий нулевой символ
  return buffer;
}

int charToInt(const char *str) {
  int result = 0;
  int sign = 1;
  int i = 0;

  // Преобразование каждого символа в цифру и добавление к результату
  while (str[i] != '\0') {
    result = result * 10 + (str[i] - '0');
    i++;
  }

  return sign * result;
}

void printString(std::ofstream &output, const char *str) {
  int len = length(str); // Используем функцию strlen для определения длины строки
  output << "Head —> ";
  for (int i = 0; i < len; ++i) {
    output << str[i];
    if ((i + 1) % 5 == 0 && (i + 1) != len) // Проверяем, что длина строки не кратна 5 и не является последним символом
      output << "->"; // Выводим стрелку только если длина строки не кратна 5 и не последний символ
  }
  output << " -> NULL";
}
```

Functional.h

```
//
// Created by Анастасия on 12.05.2024.
//

#ifndef UNTITLED_FUNCTIONAL_H
#define UNTITLED_FUNCTIONAL_H

#include "model/list/StringList.h"
#include "model/list/IntList.h"
#include "model/list/PlayerList.h"

//1 игроков, сгруппированные по позиции и возрастной категории
void printPlayersGroupedByPositionAndAgeCategory(StringList *positionList, IntList *dateOfBirthList,
                                                 PlayerList *playerList);

//2 список игроков и кандидатов, которые играли ранее в одних командах
void printPlayersInTeams(PlayerList *playerList, StringList *teamNameList);

//3 учетные карточки на каждого игрока, упорядоченные (отдельно) по общему числу игр, голов и голевых передач за все команды,
void printPlayerCards(PlayerList *playerList, int choice);

//4 учетные карточки на каждого игрока команды, упорядоченные (отдельно) по общему числу игр, голов и голевых передач за команду
void printPlayerCardsByTeam_1(PlayerList *playerList, StringList *teamNameList);

void printPlayerCardsByTeam_2(PlayerList *playerList, StringList *teamNameList);

void printPlayerCardsByTeam_3(PlayerList *playerList, StringList *teamNameList);

void printPlayerCardsByTeam_4(PlayerList *playerList, StringList *teamNameList);

#endif //UNTITLED_FUNCTIONAL_H
```

Functional.cpp

```
//
// Created by Анастасия on 12.05.2024.
//

#include <iomanip>
#include "Functional.h"
#include "model/list/StringList.h"
#include "model/list/IntList.h"
#include "model/list/PlayerList.h"

//1 игроков, сгруппированные по позиции и возрастной категории
void printPlayersGroupedByPositionAndAgeCategory(StringList *positionList, IntList *dateOfBirthList,
                                                 PlayerList *playerList) {
  std::ofstream outputFile("output1.txt");
  if (!outputFile.is_open()) {
    std::cerr << "Unable to open output file";
    return;
  }

  auto positionNode = positionList->head;

  // Определим ширину каждого столбца
  int positionWidth = 15;
  int yearWidth = 8;
  int nameWidth = 25;

  // Флаг для отслеживания, была ли уже выведена позиция
  bool positionPrinted = false;

  // Заголовки таблицы
  outputFile << std::left << std::setw(positionWidth) << "Position" << std::setw(yearWidth) << "Year"
             << std::setw(nameWidth) << "Player"
             << std::endl;
  outputFile << std::setfill('-') << std::setw(positionWidth + yearWidth + nameWidth) << "" << std::setfill(' ')
             << std::endl;

  while (positionNode != nullptr) {
    auto dateOfBirthNode = dateOfBirthList->head;
    while (dateOfBirthNode != nullptr) {
```

```
            auto playerNode = playerList->head;
            while (playerNode != nullptr) {
                if ((*playerNode->data.position == positionNode->data) &&
                    (dateOfBirthNode->data == playerNode->data.year)) {
                    // Если позиция еще не была выведена, выводим ее
                    if (!positionPrinted) {
                        outputFile << std::left << std::setw(positionWidth) << positionNode->data
                                   << std::setw(yearWidth) << dateOfBirthNode->data
                                   << std::setw(nameWidth) << *(playerNode->data.name) << std::endl;
                        positionPrinted = true;
                    } else {
                        // Выводим только данные игрока
                        outputFile << std::left << std::setw(positionWidth) << ""
                                   << std::setw(yearWidth) << dateOfBirthNode->data
                                   << std::setw(nameWidth) << *(playerNode->data.name) << std::endl;
                    }
                }
                playerNode = playerNode->next;
            }
            dateOfBirthNode = dateOfBirthNode->next;
        }

        // Сбрасываем флаг для следующей позиции
        positionPrinted = false;
        positionNode = positionNode->next;
    }

    outputFile.close();
}

bool playedInTeam(const Player &player, const char *teamName) {
    auto *currentTeamNode = player.statList->head;
    while (currentTeamNode != nullptr) {
        if (strcmp(*currentTeamNode->data.teamName, teamName) == 0) {
            return true;
        }
        currentTeamNode = currentTeamNode->next;
    }
    return false;
}

//2 список игроков и кандидатов, которые играли ранее в одних командах
void printPlayersInTeams(PlayerList *playerList, StringList *teamNameList) {
    std::ofstream outputFile("output2.txt");
    if (!outputFile.is_open()) {
        std::cerr << "Unable to open output file";
        return;
    }
    int size = 40;
    auto *teamNameNode = teamNameList->head;
    while (teamNameNode != nullptr) {
        outputFile << "Team: " << teamNameNode->data << std::endl;

        // Вывод игроков команды
        PlayerNode *currentPlayerNode = playerList->head;
        while (currentPlayerNode != nullptr) {
            if (playedInTeam(currentPlayerNode->data, teamNameNode->data)) {
                outputFile << std::left << std::setw(size) << *(currentPlayerNode->data.name) <<
std::setw(size)
                           << *(currentPlayerNode->data.status) << std::endl;
            }
            currentPlayerNode = currentPlayerNode->next;
        }

        outputFile << "------------------" << std::endl << std::endl;
        teamNameNode = teamNameNode->next;
    }


    outputFile.close();
}

// Функция для сравнения игроков по общему числу игр
bool compareByPlayedMatches(const Player &player1, const Player &player2) {
    return player1.commonPlayedMatches > player2.commonPlayedMatches;
}

// Функция для сравнения игроков по общему числу забитых голов
bool compareByGoalsScored(const Player &player1, const Player &player2) {
    return player1.commonGoalsScored > player2.commonGoalsScored;
}

// Функция для сравнения игроков по общему числу пропущенных голов
bool compareByGoalsConceded(const Player &player1, const Player &player2) {
    return player1.commonGoalsConceded > player2.commonGoalsConceded;
}

// Функция для сравнения игроков по общему числу голевых передач
bool compareByAssists(const Player &player1, const Player &player2) {
    return player1.commonAssists > player2.commonAssists;
}

void swapPlayerNode(PlayerNode *playerNode, PlayerNode *playerNode2) {
    // Обмен id игроков
    auto tempId = playerNode->data.idPlayer;
    playerNode->data.idPlayer = playerNode2->data.idPlayer;
    playerNode2->data.idPlayer = tempId;

    // Обмен имен игроков
    auto tempName = playerNode->data.name;
    playerNode->data.name = playerNode2->data.name;
    playerNode2->data.name = tempName;

    // Обмен года рождения
    auto tempYear = playerNode->data.year;
    playerNode->data.year = playerNode2->data.year;
    playerNode2->data.year = tempYear;

    // Обмен городов
    auto tempCity = playerNode->data.city;
    playerNode->data.city = playerNode2->data.city;
    playerNode2->data.city = tempCity;

    // Обмен позиций
    auto tempPosition = playerNode->data.position;
    playerNode->data.position = playerNode2->data.position;
    playerNode2->data.position = tempPosition;

    // Обмен статусов
    auto tempStatus = playerNode->data.status;
    playerNode->data.status = playerNode2->data.status;
    playerNode2->data.status = tempStatus;

    // Обмен сыгранных матчей
    auto tempPlayedMatches = playerNode->data.commonPlayedMatches;
    playerNode->data.commonPlayedMatches = playerNode2->data.commonPlayedMatches;
    playerNode2->data.commonPlayedMatches = tempPlayedMatches;

    // Обмен забитых голов
    auto tempGoalsScored = playerNode->data.commonGoalsScored;
    playerNode->data.commonGoalsScored = playerNode2->data.commonGoalsScored;
    playerNode2->data.commonGoalsScored = tempGoalsScored;

    // Обмен пропущенных голов
    auto tempGoalsConceded = playerNode->data.commonGoalsConceded;
    playerNode->data.commonGoalsConceded                                    =
playerNode2->data.commonGoalsConceded;
    playerNode2->data.commonGoalsConceded = tempGoalsConceded;

    // Обмен голевых передач
    auto tempAssists = playerNode->data.commonAssists;
    playerNode->data.commonAssists = playerNode2->data.commonAssists;
    playerNode2->data.commonAssists = tempAssists;

    // Обмен статистики команд
    auto tempStatList = playerNode->data.statList;
    playerNode->data.statList = playerNode2->data.statList;
    playerNode2->data.statList = tempStatList;
}

void sortPlayers(PlayerList *playerList, bool (*compare)(const Player &player1, const Player
&player2)) {
    if (!playerList || !playerList->head || !compare)
        return;

    PlayerNode *current = playerList->head;
    while (current != nullptr) {
        PlayerNode *nextNode = current->next;
        while (nextNode != nullptr) {
            if (compare(nextNode->data, current->data)) {
                swapPlayerNode(current, nextNode);
            }
            nextNode = nextNode->next;
        }
        current = current->next;
    }
}

//3 учетные карточки на каждого игрока, упорядоченные (отдельно) по общему числу
игр, голов и голевых передач за все команды,
void printPlayerCards(PlayerList *playerList, int choice) {
    std::ofstream outputFile("output3.txt");
    if (!outputFile.is_open()) {
        std::cerr << "Unable to open output file";
        return;
    }

    // Определим ширину каждого столбца
    int nameWidth = 35;
    int size = 15;

    // Заголовки таблицы
    outputFile << std::left << std::setw(nameWidth) << "Player" << std::setw(size) <<
"Matches"
               << std::setw(size) << "Goals Scored" << std::setw(size) << "Goals Conceded"
               << std::setw(size) << "Assists" << std::endl;
    outputFile << std::setfill('-') << std::setw(nameWidth + 4 * size) << "" << std::setfill(' ') <<
std::endl;

    // Сортировка игроков

    switch (choice) {
        case 1:
            sortPlayers(playerList, compareByPlayedMatches);
            break;
        case 2:
            sortPlayers(playerList, compareByGoalsScored);
            break;
        case 3:
            sortPlayers(playerList, compareByGoalsConceded);
            break;
        case 4:
            sortPlayers(playerList, compareByAssists);
            break;
        case 5:
            break;
        default:
            std::cout << "Wrong choice. Try again" << std::endl;
            break;
    }

    // Вывод учетных карточек игроков
    PlayerNode *current = playerList->head;
    while (current != nullptr) {
        outputFile << std::left << std::setw(nameWidth) << *(current->data.name)
                   << std::setw(size) << current->data.commonPlayedMatches
                   << std::setw(size) << current->data.commonGoalsScored
                   << std::setw(size) << current->data.commonGoalsConceded
                   << std::setw(size) << current->data.commonAssists << std::endl;
        current = current->next;
    }

    outputFile << std::endl;
    outputFile.close();
}

bool compareBy_PlayedMatches(const Player &player1, const Player &player2, const char
*teamName) {
    int player_1 = 0;
    int player_2 = 0;

    auto teamStatNode1 = player1.statList->head;
    while (teamStatNode1 != nullptr) {
        if (strcmp(*teamStatNode1->data.teamName, teamName) == 0) {
            player_1 = teamStatNode1->data.playedMatches;
            break;
        }
        teamStatNode1 = teamStatNode1->next;
    }

    auto teamStatNode2 = player2.statList->head;
    while (teamStatNode2 != nullptr) {
        if (strcmp(*teamStatNode2->data.teamName, teamName) == 0) {
            player_2 = teamStatNode2->data.playedMatches;
            break;
        }
        teamStatNode2 = teamStatNode2->next;
```

```
    }

    return player_1 > player_2;
}

bool compareBy_GoalsScored(const Player &player1, const Player &player2, const char *teamName) {
    int player_1 = 0;
    int player_2 = 0;

    auto teamStatNode1 = player1.statList->head;
    while (teamStatNode1 != nullptr) {
        if (strcmp(*teamStatNode1->data.teamName, teamName) == 0) {
            player_1 = teamStatNode1->data.goalsScored;
            break;
        }
        teamStatNode1 = teamStatNode1->next;
    }

    auto teamStatNode2 = player2.statList->head;
    while (teamStatNode2 != nullptr) {
        if (strcmp(*teamStatNode2->data.teamName, teamName) == 0) {
            player_2 = teamStatNode2->data.goalsScored;
            break;
        }
        teamStatNode2 = teamStatNode2->next;
    }

    return player_1 > player_2;
}

bool compareBy_GoalsConceded(const Player &player1, const Player &player2, const char *teamName) {
    int player_1 = 0;
    int player_2 = 0;

    auto teamStatNode1 = player1.statList->head;
    while (teamStatNode1 != nullptr) {
        if (strcmp(*teamStatNode1->data.teamName, teamName) == 0) {
            player_1 = teamStatNode1->data.goalsConceded;
            break;
        }
        teamStatNode1 = teamStatNode1->next;
    }

    auto teamStatNode2 = player2.statList->head;
    while (teamStatNode2 != nullptr) {
        if (strcmp(*teamStatNode2->data.teamName, teamName) == 0) {
            player_2 = teamStatNode2->data.goalsConceded;
            break;
        }
        teamStatNode2 = teamStatNode2->next;
    }

    return player_1 > player_2;
}

bool compareBy_Assists(const Player &player1, const Player &player2, const char *teamName) {
    int player_1 = 0;
    int player_2 = 0;

    auto teamStatNode1 = player1.statList->head;
    while (teamStatNode1 != nullptr) {
        if (strcmp(*teamStatNode1->data.teamName, teamName) == 0) {
            player_1 = teamStatNode1->data.assists;
            break;
        }
        teamStatNode1 = teamStatNode1->next;
    }

    auto teamStatNode2 = player2.statList->head;
    while (teamStatNode2 != nullptr) {
        if (strcmp(*teamStatNode2->data.teamName, teamName) == 0) {
            player_2 = teamStatNode2->data.assists;
            break;
        }
        teamStatNode2 = teamStatNode2->next;
    }

    return player_1 > player_2;
}

void sortPlayersByTeamName(PlayerList *playerList, const char *teamName,
                           bool (*compare)(const Player &player1, const Player &player2, const char *teamName)) {
    if (!playerList || !playerList->head || !compare) return;

    for (PlayerNode *i = playerList->head; i != nullptr; i = i->next) {
        for (PlayerNode *j = i->next; j != nullptr; j = j->next) {
            if (compare(j->data, i->data, teamName)) {
                swapPlayerNode(i, j);
            }
        }
    }
}

void printPlayerStats(std::ofstream &outputFile, PlayerNode *current, const char *teamName, int nameWidth, int size) {
    while (current != nullptr) {
        if (playedInTeam(current->data, teamName)) {
            auto teamStatNode = current->data.statList->head;
            while (teamStatNode != nullptr) {
                if (*teamStatNode->data.teamName == teamName) {
                    outputFile << std::left << std::setw(nameWidth) << *(current->data.name)
                               << std::setw(size) << teamStatNode->data.playedMatches
                               << std::setw(size) << teamStatNode->data.goalsScored
                               << std::setw(size) << teamStatNode->data.goalsConceded
                               << std::setw(size) << teamStatNode->data.assists << std::endl;
                }
                teamStatNode = teamStatNode->next;
            }
        }
        current = current->next;
    }
}

void printPlayerCardsByTeam_1(PlayerList *playerList, StringList *teamNameList) {
    if (!playerList || !playerList->head) {
        std::cerr << "Player list is empty!" << std::endl;
        return;
    }

    std::ofstream outputFile("output4.1.txt");
    if (!outputFile.is_open()) {
        std::cerr << "Unable to open output file" << std::endl;
        return;
    }

    int nameWidth = 35;
    int size = 15;

    auto teamNameNode = teamNameList->head;
    while (teamNameNode != nullptr) {
        // Вывод учетных карточек игроков по команде
        outputFile << "Player Cards for Team: " << teamNameNode->data << std::endl;
        outputFile << std::left << std::setw(nameWidth) << "Player"
                   << std::setw(size) << "Matches"
                   << std::setw(size) << "GoalsScored"
                   << std::setw(size) << "GoalsConceded"
                   << std::setw(size) << "Assists" << std::endl;
        sortPlayersByTeamName(playerList, teamNameNode->data, compareBy_PlayedMatches);

        PlayerNode *current = playerList->head;
        printPlayerStats(outputFile, current, teamNameNode->data, nameWidth, size);

        teamNameNode = teamNameNode->next;
        outputFile << "----------------------------------------" << std::endl;
    }

    outputFile.close();
}

void printPlayerCardsByTeam_2(PlayerList *playerList, StringList *teamNameList) {
    if (!playerList || !playerList->head) {
        std::cerr << "Player list is empty!" << std::endl;
        return;
    }

    std::ofstream outputFile("output4.2.txt");
    if (!outputFile.is_open()) {
        std::cerr << "Unable to open output file" << std::endl;
        return;
    }

    int nameWidth = 35;
    int size = 15;

    auto teamNameNode = teamNameList->head;
    while (teamNameNode != nullptr) {
        // Вывод учетных карточек игроков по команде
        outputFile << "Player Cards for Team: " << teamNameNode->data << std::endl;
        outputFile << std::left << std::setw(nameWidth) << "Player"
                   << std::setw(size) << "Matches"
                   << std::setw(size) << "GoalsScored"
                   << std::setw(size) << "GoalsConceded"
                   << std::setw(size) << "Assists" << std::endl;
        sortPlayersByTeamName(playerList, teamNameNode->data, compareBy_GoalsScored);

        PlayerNode *current = playerList->head;
        printPlayerStats(outputFile, current, teamNameNode->data, nameWidth, size);

        teamNameNode = teamNameNode->next;
        outputFile << "----------------------------------------" << std::endl;
    }

    outputFile.close();
}

void printPlayerCardsByTeam_3(PlayerList *playerList, StringList *teamNameList) {
    if (!playerList || !playerList->head) {
        std::cerr << "Player list is empty!" << std::endl;
        return;
    }

    std::ofstream outputFile("output4.3.txt");
    if (!outputFile.is_open()) {
        std::cerr << "Unable to open output file" << std::endl;
        return;
    }

    int nameWidth = 35;
    int size = 15;

    auto teamNameNode = teamNameList->head;
    while (teamNameNode != nullptr) {
        // Вывод учетных карточек игроков по команде
        outputFile << "Player Cards for Team: " << teamNameNode->data << std::endl;
        outputFile << std::left << std::setw(nameWidth) << "Player"
                   << std::setw(size) << "Matches"
                   << std::setw(size) << "GoalsScored"
                   << std::setw(size) << "GoalsConceded"
                   << std::setw(size) << "Assists" << std::endl;
        sortPlayersByTeamName(playerList, teamNameNode->data, compareBy_GoalsConceded);
        PlayerNode *current = playerList->head;
        printPlayerStats(outputFile, current, teamNameNode->data, nameWidth, size);

        teamNameNode = teamNameNode->next;
        outputFile << "----------------------------------------" << std::endl;
    }

    outputFile.close();
}

void printPlayerCardsByTeam_4(PlayerList *playerList, StringList *teamNameList) {
    if (!playerList || !playerList->head) {
        std::cerr << "Player list is empty!" << std::endl;
        return;
    }

    std::ofstream outputFile("output4.4.txt");
    if (!outputFile.is_open()) {
        std::cerr << "Unable to open output file" << std::endl;
        return;
    }

    int nameWidth = 35;
    int size = 15;

    auto teamNameNode = teamNameList->head;
    while (teamNameNode != nullptr) {
        // Вывод учетных карточек игроков по команде
        outputFile << "Player Cards for Team: " << teamNameNode->data << std::endl;
        outputFile << std::left << std::setw(nameWidth) << "Player"
                   << std::setw(size) << "Matches"
```

```
                << std::setw(size) << "GoalsScored"
                << std::setw(size) << "GoalsConceded"
                << std::setw(size) << "Assists" << std::endl;
        sortPlayersByTeamName(playerList, teamNameNode->data, compareBy_Assists);
        PlayerNode *current = playerList->head;
        printPlayerStats(outputFile, current, teamNameNode->data, nameWidth, size);

        teamNameNode = teamNameNode->next;
        outputFile << "----------------------------------------" << std::endl;
    }

    outputFile.close();
}
```

Player

```
//
// Created by Анастасия on 01.05.2024.
//

#ifndef PROGA_KURS_PLAYER_H
#define PROGA_KURS_PLAYER_H

#include "../list/TeamStatList.h"

struct Player {


    Player(int id, char **playerName, int playerYear, char **playerCity, char **playerPosition, char **playerStatus);

    int idPlayer;
    char **name;
    int year;
    char **city;
    char **position;
    char **status;

    TeamStatList *statList;

    int commonPlayedMatches = 0;     // Сыгранные матчи
    int commonGoalsScored = 0;       // Забитые голы
    int commonGoalsConceded = 0;     // Пропущенные голы
    int commonAssists = 0;           // Голевые передачи

    void appendTeamStat(const TeamStat& value);

    ~Player();

    friend std::ostream &operator<<(std::ostream &os, const Player &player);

    friend bool operator==(const Player &lhs, const Player &rhs);

};

#endif //PROGA_KURS_PLAYER_H
```

Player

```
#include "Player.h"

Player::Player(
        int id,
        char **playerName,
        int playerYear,
        char **playerCity,
        char **playerPosition,
        char **playerStatus
) : idPlayer(id), name(playerName), year(playerYear), city(playerCity), position(playerPosition), status(playerStatus) {
    this->statList = new TeamStatList();
}


Player::~Player() = default;

void Player::appendTeamStat(const TeamStat& value) {
    this->statList->appendNode(value);
    commonPlayedMatches += value.playedMatches;     // Сыгранные матчи
    commonGoalsScored += value.goalsScored;         // Забитые голы
    commonGoalsConceded += value.goalsConceded;     // Пропущенные голы
    commonAssists += value.assists;                 // Голевые передачи
}

std::ostream &operator<<(std::ostream &os, const Player &player) {
    os << player.idPlayer << " ";
    os << *(player.name) << " ";
    os << player.year << " ";
    os << *(player.city) << " ";
    os << *(player.position) << " ";
    os << *(player.status) << " ";
    os << player.commonPlayedMatches << " ";
    auto* statListNode = player.statList;
    auto* current = statListNode->head;
    while (current != nullptr) {
        os << *(current->data.teamName) << " ";
        current = current->next;
    }
    return os;
}

bool operator==(const Player &lhs, const Player &rhs) {
    // Сравниваем все поля структуры Player
    return lhs.idPlayer == rhs.idPlayer;
}
```

TeamStat

```
#ifndef PROGA_KURS_TEAMSTAT_H
#define PROGA_KURS_TEAMSTAT_H

#include "../../Util.h" // Включаем заголовочный файл с определениями функций length(), copy() и cleanup()
#include <iostream> // Для использования объекта потока std::cout

struct TeamStat {

    char **teamName;
    int playedMatches = 0;     // Сыгранные матчи
    int goalsScored = 0;       // Забитые голы
    int goalsConceded = 0;     // Пропущенные голы
    int assists = 0;           // Голевые передачи
    // Конструктор
    TeamStat(char **teamName, int matches, int scored, int conceded, int assist);

    // Деструктор
    ~TeamStat();

    friend std::ostream &operator<<(std::ostream &os, const TeamStat &name);

    friend bool operator==(const TeamStat &name1, const TeamStat &name2);
};
```

```
#endif //PROGA_KURS_TEAMSTAT_H
```

TeamStat

```
#include <iostream>
#include "TeamStat.h"

// Конструктор
TeamStat::TeamStat(char **teamName, int matches, int scored, int conceded, int assist) :
        teamName(teamName),
        playedMatches(matches),
        goalsScored(scored),
        goalsConceded(conceded),
        assists(assist) {}

// Деструктор
TeamStat::~TeamStat() {
    delete[]teamName;
}

std::ostream &operator<<(std::ostream &os, const TeamStat &stat) {
    os << *stat.teamName << " " << stat.playedMatches << " " << stat.goalsScored << " " << stat.goalsConceded << " "
       << stat.assists;
    return os;
}

bool operator==(const TeamStat &name1, const TeamStat &name2) {
    return strcmp(*name1.teamName, *name2.teamName) == 0;
}
```

IntList

```
#ifndef UNTITLED_INTLIST_H
#define UNTITLED_INTLIST_H

#include "../node/IntNode.h"

// Класс для списка int
struct IntList {

    IntNode *head; // Указатель на начало списка
    IntNode *tail; // Указатель на конец списка

    // Конструктор по умолчанию
    IntList();

    // Деструктор
    ~IntList();

    // Добавление элемента в конец списка, если его там нет
    void append(int value);

    // Перегрузка оператора вывода для вывода списка в поток
    friend std::ostream &operator<<(std::ostream &os, const IntList &list);
};

#endif //UNTITLED_INTLIST_H
```

IntList

```
#include <ostream>
#include "IntList.h"

// Конструктор по умолчанию
IntList::IntList() : head(nullptr), tail(nullptr) {}

// Деструктор
IntList::~IntList() {
    // Удаление всех узлов списка
    IntNode *current = head;
    while (current != nullptr) {
        IntNode *temp = current;
        current = current->next;
        delete temp;
    }
}

// Добавление элемента в конец списка, если его там нет
void IntList::append(int value) {
    // Проверка наличия элемента в списке
    IntNode *current = head;
    while (current != nullptr) {
        if (current->data == value) {
            // Элемент уже существует в списке, выходим из функции
            return;
        }
        current = current->next;
    }

    // Создание нового узла
    auto* newNode = new IntNode(value);

    // Если список пуст или новый элемент меньше первого элемента, вставляем его в начало
    if (head == nullptr || value < head->data) {
        newNode->next = head;
        head = newNode;
        if (tail == nullptr) {
            tail = head; // Если список был пуст, новый элемент также становится хвостом
        }
        return;
    }

    // Поиск места для вставки нового элемента
    current = head;
    while (current->next != nullptr && value > current->next->data) {
        current = current->next;
    }

    // Вставка нового элемента после текущего узла
    newNode->next = current->next;
    current->next = newNode;

    // Если новый элемент добавлен в конец списка, обновляем указатель на хвост
    if (current == tail) {
        tail = newNode;
    }
}

// Перегрузка оператора вывода для вывода списка в поток
std::ostream& operator<<(std::ostream& os, const IntList& list) {
    IntNode* current = list.head;
    while (current != nullptr) {
        os << current->data;
        if (current->next != nullptr) {
            os << "\t"; // Выводим табуляцию между элементами списка
        }
        current = current->next;
    }
    return os;
```

	} #endif //UNTITLED_STRINGLIST_H

StringList	#ifndef UNTITLED_STRINGLIST_H
	#define UNTITLED_STRINGLIST_H
	
	#include "../node/StringNode.h"
	
	// Класс для списка char*
	struct StringList {
	StringNode* head; // Указатель на начало списка
	StringNode* tail; // Указатель на конец списка
	
	// Конструктор по умолчанию
	StringList();
	
	// Деструктор
	~StringList();
	
	// Добавление элемента в конец списка, если его там нет
	StringNode* append(const char* value);
	
	// Перегрузка оператора вывода для вывода списка в поток
	friend std::ostream& operator<<(std::ostream& os, const StringList& list);
	
	void print(std::ofstream &output);
	};
	
	#endif //UNTITLED_STRINGLIST_H

StringList	#include "StringList.h"
	
	// Конструктор по умолчанию
	StringList::StringList() : head(nullptr), tail(nullptr) {}
	
	// Деструктор
	StringList::~StringList() {
	// Удаление всех узлов списка
	StringNode *current = head;
	while (current != nullptr) {
	StringNode *temp = current;
	current = current->next;
	delete temp;
	}
	}
	
	// Добавление элемента в конец списка, если его там нет
	StringNode *StringList::append(const char *value) {
	// Проверка наличия элемента в списке
	StringNode *current = head;
	while (current != nullptr) {
	if (strcmp(current->data, value) == 0) {
	// Элемент уже существует в списке, выходим из функции
	return current;
	}
	current = current->next;
	}
	
	// Создание нового узла
	auto* newNode = new StringNode(value);
	
	// Если список пуст или новый элемент меньше первого элемента, вставляем его в начало
	if (head == nullptr || strcmp(value, head->data) < 0) {
	newNode->next = head;
	head = newNode;
	if (tail == nullptr) {
	tail = head; // Если список был пуст, новый элемент также становится хвостом
	}
	return newNode;
	}
	
	// Поиск места для вставки нового элемента
	current = head;
	while (current->next != nullptr && strcmp(value, current->next->data) > 0) {
	current = current->next;
	}
	
	// Вставка нового элемента после текущего узла
	newNode->next = current->next;
	current->next = newNode;
	
	// Если новый элемент добавлен в конец списка, обновляем указатель на хвост
	if (current == tail) {
	tail = newNode;
	}
	return newNode;
	}
	
	// Перегрузка оператора вывода для вывода списка в поток
	std::ostream& operator<<(std::ostream& os, const StringList& list) {
	StringNode* current = list.head;
	while (current != nullptr) {
	os << current->data;
	if (current->next != nullptr) {
	os << "\t"; // Выводим табуляцию между элементами списка
	}
	current = current->next;
	}
	return os;
	}
	
	/*void StringList::print(std::ofstream &output){
	StringNode* current = this->head;
	while (current != nullptr) {
	printString(output,current->data);
	if (current->next != nullptr) {
	output << "\t"; // Выводим табуляцию между элементами списка
	}
	current = current->next;
	}
	}
	*/

PlayerList	#ifndef PROGA_KURS_PLAYERLIST_H
	#define PROGA_KURS_PLAYERLIST_H
	
	#include <iostream>
	#include "../node/PlayerNode.h"
	
	struct PlayerList {
	PlayerNode *head; // Указатель на начало списка
	PlayerNode *tail; // Указатель на конец списка
	
	// Конструктор по умолчанию
	PlayerList();
	
	// Деструктор для освобождения памяти

	~PlayerList();
	
	// Добавление узла в конец списка
	void appendNode(const Player& value);
	
	// Оператор вывода узла в поток
	friend std::ostream &operator<<(std::ostream &os, const PlayerList &list);
	
	[[nodiscard]] Player *findById(int playerId) const;
	
	static void printPlayer(const PlayerList &list);
	};
	
	#endif //PROGA_KURS_PLAYERLIST_H

PlayerList	#include "PlayerList.h"
	
	PlayerList::PlayerList() : head(nullptr), tail(nullptr) {}
	
	// Деструктор для освобождения памяти
	PlayerList::~PlayerList() {
	PlayerNode *current = head; // Указатель на текущий узел
	
	while (current != nullptr) {
	PlayerNode *next = current->next; // Сохраняем указатель на следующий узел
	delete current; // Освобождаем память для текущего узла
	current = next; // Переходим к следующему узлу
	}
	
	head = nullptr; // Устанавливаем указатель на начало списка в nullptr
	tail = nullptr; // Устанавливаем указатель на конец списка в nullptr
	}
	
	// Добавление узла в конец списка
	void PlayerList::appendNode(const Player &value) {
	
	// Проверяем, существует ли уже такое значение в списке
	PlayerNode *current = head;
	while (current != nullptr) {
	if (current->data == value) {
	// Значение уже присутствует в списке, поэтому выходим из функции
	return;
	}
	current = current->next;
	}
	
	// Создаем новый узел с заданным значением
	auto *newNode = new PlayerNode(value);
	
	// Если список пуст, устанавливаем новый узел как начало и конец списка
	if (head == nullptr) {
	head = newNode;
	tail = newNode;
	} else {
	// Добавляем новый узел в конец списка и обновляем указатель на хвост
	tail->next = newNode;
	tail = newNode;
	}
	}
	
	Player *PlayerList::findById(int playerId) const {
	auto currentNode = this->head;
	// Начинаем с головы списка
	while (currentNode != nullptr) {
	if (currentNode->data.idPlayer == playerId) {
	return &(currentNode->data); // Возвращаем указатель на игрока, если найден
	}
	currentNode = currentNode->next; // Переходим к следующему узлу
	}
	return nullptr; // Если игрок с указанным идентификатором не найден, возвращаем nullptr
	}
	
	// Оператор вывода узла в поток
	std::ostream &operator<<(std::ostream &os, const PlayerList &list) {
	PlayerNode *current = list.head;
	os<< "Head NULL-> ";
	while (current != nullptr) {
	os << current->data << " -> ";
	current = current->next;
	}
	os<<"NULL Tail\n" ;
	return os;
	}
	
	void PlayerList::printPlayer(const PlayerList &list) {
	std::ofstream output("output_player.txt");
	if (!output.is_open()) {
	std::cerr << "Error: Unable to open output file." << std::endl;
	return;
	}
	PlayerNode *current = list.head;
	output << "Head NULL" << "\n";
	output << "----------" << std::endl;
	output << " | /|\" << "\n";
	output << "\\|/ |" << "\n";
	while (current != nullptr) {
	output << "----------" << std::endl;
	output << "id:\t " << current->data.idPlayer << std::endl;
	output << "name:\t ";
	printString(output,*current->data.name);
	output<<std::endl;
	output << "year:\t " <<current->data.year << std::endl;
	output << "city:\t ";
	printString(output,*current->data.city);
	output<<std::endl;
	output << "position:\t ";
	printString(output,*current->data.position);
	output<<std::endl;
	output << "status:\t ";
	printString(output,*current->data.status);
	output<<std::endl;
	output << "teamName:\t ";
	
	Player currentPlayer = current->data;
	auto* statList = currentPlayer.statList;
	auto* teamNode = statList->head;
	
	while (teamNode != nullptr) {
	printString(output, *teamNode->data.teamName);
	output << " "; // Разделяем команды пробелом
	teamNode = teamNode->next;
	}
	//printString(output,*current->data.statList->head->data.teamName);

```
        output<<std::endl;
        output << "commonPlayedMatches:\t " <<current->data.commonPlayedMatches << std::endl;
        output << "commonGoalsScored:\t " <<current->data.commonGoalsScored << std::endl;
        output << "commonGoalsConceded:\t " <<current->data.commonGoalsConceded << std::endl;
        output << "commonAssists:\t " <<current->data.commonAssists << std::endl;
        output << "----------------------------------------" << std::endl;
        output << " |/|\\" << "\n";
        output << "\\|/ |" << "\n";
        current = current->next;
    }
    output << "----------------------------------------" << std::endl;
    output << "NULL Tail" << "\n";
}
```

TeamStatList

```
#ifndef PROGA_KURS_TEAMSTATLIST_H
#define PROGA_KURS_TEAMSTATLIST_H

#include <iostream>
#include "../node/TeamStatNode.h"

struct TeamStatList {
    TeamStatNode *head; // Указатель на начало списка
    TeamStatNode *tail; // Указатель на конец списка

    // Конструктор по умолчанию
    TeamStatList();

    // Деструктор для освобождения памяти
    ~TeamStatList();

    // Добавление узла в конец списка
    void appendNode(const TeamStat& value);

    // Оператор вывода узла в поток
    friend std::ostream &operator<<(std::ostream &os, const TeamStatList &list);
};

#endif //PROGA_KURS_TEAMSTATLIST_H
```

TeamStatList

```
#include "TeamStatList.h"

TeamStatList::TeamStatList() : head(nullptr), tail(nullptr) {}


// Деструктор для освобождения памяти
TeamStatList::~TeamStatList() {
    TeamStatNode *current = head; // Указатель на текущий узел

    while (current != nullptr) {
        TeamStatNode *next = current->next; // Сохраняем указатель на следующий узел
        delete current; // Освобождаем память для текущего узла
        current = next; // Переходим к следующему узлу
    }

    head = nullptr; // Устанавливаем указатель на начало списка в nullptr
    tail = nullptr; // Устанавливаем указатель на конец списка в nullptr
}

// Добавление узла в конец списка
void TeamStatList::appendNode(const TeamStat &value) {
    // Проверяем, существует ли уже такое значение в списке
    TeamStatNode *current = head;
    while (current != nullptr) {
        if (current->data == value) {
            // Значение уже присутствует в списке, поэтому выходим из функции
            return;
        }
        current = current->next;
    }
    // Создаем новый узел с заданным значением
    auto *newNode = new TeamStatNode(value);

    // Если список пуст, устанавливаем новый узел как начало и конец списка
    if (head == nullptr) {
        head = newNode;
        tail = newNode;
    } else {
        // Добавляем новый узел в конец списка и обновляем указатель на хвост
        tail->next = newNode;
        tail = newNode;
    }
}

// Оператор вывода узла в поток
std::ostream &operator<<(std::ostream &os, const TeamStatList &list) {
    TeamStatNode *current = list.head;
    while (current != nullptr) {
        os << current->data << " -> ";
        current = current->next;
    }
    os << "NULL";
    return os;
}
```

IntNode

```
#ifndef UNTITLED_INTNODE_H
#define UNTITLED_INTNODE_H

struct IntNode {
    int data; // Данные узла
    IntNode* next; // Указатель на следующий узел

    // Конструктор
    explicit IntNode(int value);

    // Деструктор
    ~IntNode();
};

#endif //UNTITLED_INTNODE_H
```

IntNode

```
#include "IntNode.h"

// Конструктор
IntNode::IntNode(int value) : data(value), next(nullptr) {}

// Деструктор
IntNode::~IntNode() = default;
```

StringNode

```
#ifndef UNTITLED_STRINGNODE_H
#define UNTITLED_STRINGNODE_H

#include "../../Util.h"

struct StringNode {
    char* data; // Указатель на данные
    StringNode* next; // Указатель на следующий узел

    // Конструктор
    explicit StringNode(const char* value);

    ~StringNode();
};

#endif //UNTITLED_STRINGNODE_H
```

StringNode

```
#include "StringNode.h"
// Конструктор
StringNode::StringNode(const char* value) : next(nullptr) {
    // Выделение памяти для копии строки и копирование данных
    int len = length(value);
    data = new char[len + 1];
    copy(value, data, len);
}

// Деструктор для освобождения памяти
StringNode::~StringNode() {
    delete[] data; // Освобождаем память, выделенную для данных
}

// Оператор сравнения равенства NameNode
bool operator==(const StringNode &lhs, const StringNode &rhs) {
    // Сравниваем данные игроков
    return strcmp(lhs.data, rhs.data) == 0;
}
```

PlayerNode

```
#ifndef PROGA_KURS_PLAYERNODE_H
#define PROGA_KURS_PLAYERNODE_H

#include <iostream>
#include "../core/Player.h"

struct PlayerNode {
    Player data; // Исправлено на Position
    PlayerNode *next; // Указатель на следующий узел

    // Конструктор для инициализации узла с заданным значением и указателем на следующий узел
    explicit PlayerNode(const Player& value);

    // Деструктор для освобождения ресурсов
    ~PlayerNode();

    // Оператор сравнения равенства PositionNode
    friend bool operator==(const PlayerNode &lhs, const PlayerNode &rhs);

    // Вывод узла в поток
    friend std::ostream &operator<<(std::ostream &os, const PlayerNode &node);
};

#endif //PROGA_KURS_POSITIONNODE_H
```

PlayerNode

```
#include "PlayerNode.h"

// Конструктор для инициализации узла с заданным значением и указателем на следующий узел
PlayerNode::PlayerNode(const Player &value) : data(value), next(nullptr) {}

// Деструктор для освобождения ресурсов
PlayerNode::~PlayerNode() = default;

// Оператор сравнения равенства NameNode
bool operator==(const PlayerNode &lhs, const PlayerNode &rhs) {
    // Сравниваем данные игроков
    return rhs.data.idPlayer == lhs.data.idPlayer;
}

// Вывод узла в поток
std::ostream &operator<<(std::ostream &os, const PlayerNode &node) {
    os << node.data.idPlayer << " " << *node.data.name;
    return os;
}
```

TeamStatNode

```
#ifndef PROGA_KURS_TEAMSTATNODE_H
#define PROGA_KURS_TEAMSTATNODE_H

#include "../core/TeamStat.h" // Подключаем заголовочный файл для структуры TeamName

struct TeamStatNode {
    TeamStat data;
    TeamStatNode *next;

    // Конструктор для инициализации узла с заданным значением и указателем на следующий узел
    explicit TeamStatNode(const TeamStat& value);

    // Деструктор для освобождения ресурсов
    ~TeamStatNode();
};

// Оператор сравнения равенства NameNode
bool operator==(const TeamStatNode &lhs, const TeamStatNode &rhs);

// Вывод узла в поток
std::ostream &operator<<(std::ostream &os, const TeamStatNode &node);


#endif //PROGA_KURS_TEAMSTATNODE_H
```

TeamStatNode

```
#include "TeamStatNode.h"

// Конструктор для инициализации узла с заданным значением и указателем на следующий узел
TeamStatNode::TeamStatNode(const TeamStat& value) : data(value), next(nullptr) {}

// Деструктор для освобождения ресурсов
TeamStatNode::~TeamStatNode() = default;

// Оператор сравнения равенства NameNode
bool operator==(const TeamStatNode &lhs, const TeamStatNode &rhs) {
    // Сравниваем данные игроков
    return lhs.data == rhs.data;
}

// Вывод узла в поток
std::ostream &operator<<(std::ostream &os, const TeamStatNode &node) {
    os << node.data;
    return os;
}
```

Результаты работы программы

Входные данные

```
1, Kosarevsky Pyotr Ilyich, 1999-05-11, Dushanbe, Goalkeeper, Team member
2, Kraikov Timur Viktorovich, 2001-05-02, Minsk, Goalkeeper, Team member
3, Fedorov Artem Anatolyevich, 1984-12-16, Moscow, Goalkeeper, Team member
4, Abazov Astemir Valeryevich, 1996-12-08, St. Petersburg, Defender, Team member
5, Gudkov Yan Mikhailovich, 2002-03-05, Saratov, Defender, Team member
6, Dudiev Aslan Muratovich, 1990-06-15, Dushanbe, Defender, Team member
7, Evdokimov Andrey Vladimirovich, 1999-03-09, Minsk, Defender, Team member
8, Mankov Nikita Sergeyevich, 2000-06-20, Moscow, Defender, Team member
9, Molodtsov Artem Igorevich, 1990-10-08, St. Petersburg, Defender, Team member
```

```
1, Istiklol, 50, 35, 20, 15
2, Torpedo, 30, 2, 25, 5
3, Spartak, 60, 12, 30, 20
4, Zenit, 40, 0, 35, 2
5, Sokol, 45, 28, 18, 10
5, Zenit, 15, 8, 7, 5
6, Torpedo, 55, 18, 25, 12
6, Spartak, 25, 5, 10, 7
7, Torpedo, 20, 3, 18, 4
8, Spartak, 35, 20, 12, 8
9, Sokol, 42, 22, 22, 13
9, Torpedo, 20, 3, 18, 4
```

Меню в консоли

```
 Select a function:
1. list of players grouped by position and age category
2. list of players and candidates who have previously played in the same teams
3. scorecards for each player, ordered (separately) by the total number of games, goals and assists for all teams
4. registration cards for each player of the team, ordered (separately) by the total number of games, goals and assists
for the team
5. exit
Your choice:2
 Select a function:
1. list of players grouped by position and age category
2. list of players and candidates who have previously played in the same teams
3. scorecards for each player, ordered (separately) by the total number of games, goals and assists for all teams
4. registration cards for each player of the team, ordered (separately) by the total number of games, goals and assists
for the team
5. exit
Your choice:3
 1. comparison of players by total number of matches
2. comparison of players by total number of goals scored
3. comparison of players by the total number of goals conceded
4. comparison of players by total number of assists
5. [Back to main menu]
Enter your choice:1
```

Файл с выводом 1 запроса

Position	Year	Player
Defender	1990	Dudiev Aslan Muratovich
	1990	Molodtsov Artem Igorevich
	1991	Pliev Zaurbek Igorevich
	1992	Karaev Maxim Irizalievich
	1995	Ozmanov David Tengizovich
	1995	Serikov Ilya Andreevich
	1996	Abazov Astemir Valeryevich
	1999	Evdokimov Andrey Vladimirovich
	2000	Mankov Nikita Sergeyevich
	2002	Gudkov Yan Mikhailovich
	2002	Tolstopyatov Nikolay Valeryevich
Forward	1983	Tursunov Igor Erkindzhanovich
	1987	Ezhkov Danila Andreevich
	1998	Maximenko Artem Sergeyevich
	2001	Morozov Vladislav Markovich
	2002	Pogosyan Albert Azatovich
	2003	Ivakhnov Matvey Andreevich
Goalkeeper	1984	Fedorov Artem Anatolyevich
	1999	Kosarevsky Pyotr Ilyich
	2001	Kraikov Timur Viktorovich
Midfielder	1994	Azarov Vladimir Alexeevich
	1994	Leontiev Igor Olegovich
	1995	Zabelin Pavel Vitalyevich
	1995	Weber Dmitry Grigorievich
	1996	Babaev Vladlen Igorevich
	1996	Sasin Dmitry Andreevich
	1997	Kamyshov Ilya Vladislavovich
	1997	Kozlovsky Nikita Andreevich
	1998	Yugaldin Dmitry Ilyich
	2000	Kazantsev Danil Antonovich
	2001	Maltsev Mikhail Dmitrievich
	2002	Batyrev Amir Renatovich
	2002	Samko Vladislav Igorevich

Файл с выводом 2 запроса

```
Team: Istiklol
Kosarevsky Pyotr Ilyich                 Team member
Ozmanov David Tengizovich               Team member
Babaev Vladlen Igorevich                Team member
Zabelin Pavel Vitalyevich               Team member
Leontiev Igor Olegovich                 Team member
Maltsev Mikhail Dmitrievich             Team member
Maximenko Artem Sergeyevich             Team member
Kazantsev Danil Antonovich              Candidate
Tursunov Igor Erkindzhanovich           Candidate
--------------------

Team: Sokol
Gudkov Yan Mikhailovich                 Team member
Molodtsov Artem Igorevich               Team member
Azarov Vladimir Alexeevich              Team member
Zabelin Pavel Vitalyevich               Team member
Kamyshov Ilya Vladislavovich            Team member
Kozlovsky Nikita Andreevich             Team member
Ivakhnov Matvey Andreevich              Team member
Ezhkov Danila Andreevich                Candidate
--------------------
```

Файл с выводом 3 запроса

Player	Matches	Goals Scored	Goals Conceded	Assists
Maltsev Mikhail Dmitrievich	147	85	75	58
Kazantsev Danil Antonovich	143	83	78	58
Tursunov Igor Erkindzhanovich	142	75	85	55
Maximenko Artem Sergeyevich	140	79	76	56
Zabelin Pavel Vitalyevich	122	72	61	39
Azarov Vladimir Alexeevich	90	43	49	27
Kamyshov Ilya Vladislavovich	85	52	36	25
Dudiev Aslan Muratovich	80	23	35	19
Serikov Ilya Andreevich	76	28	57	23
Molodtsov Artem Igorevich	62	25	40	17
Fedorov Artem Anatolyevich	60	12	30	20
Gudkov Yan Mikhailovich	60	36	25	15
Karaev Maxim Irizalievich	55	30	35	20
Leontiev Igor Olegovich	55	40	28	25
Babaev Vladlen Igorevich	52	38	23	22
Pogosyan Albert Azatovich	50	28	22	18
Kosarevsky Pyotr Ilyich	50	35	20	15
Samko Vladislav Igorevich	48	20	26	15
Ozmanov David Tengizovich	48	30	19	18
Kozlovsky Nikita Andreevich	48	32	25	20
Morozov Vladislav Markovich	47	25	27	14
Yugaldin Dmitry Ilyich	45	25	20	18
Batyrev Amir Renatovich	43	18	29	12
Sasin Dmitry Andreevich	42	30	22	17
Ezhkov Danila Andreevich	42	20	19	14
Abazov Astemir Valeryevich	40	0	35	2
Pliev Zaurbek Igorevich	38	15	30	9
Ivakhnov Matvey Andreevich	37	18	16	12
Weber Dmitry Grigorievich	35	18	15	10
Mankov Nikita Sergeyevich	35	20	12	8
Kraikov Timur Viktorovich	30	2	25	5
Tolstopyatov Nikolay Valeryevich	29	8	15	6
Evdokimov Andrey Vladimirovich	20	3	18	4

Файл с выводом 4 запроса

Сортировка по 1 столбцу

```
Player Cards for Team: Istiklol
Player                          Matches         GoalsScored     GoalsConceded   Assists
Leontiev Igor Olegovich         55              40              28              25
Zabelin Pavel Vitalyevich       52              38              23              22
Babaev Vladlen Igorevich        52              38              23              22
Maltsev Mikhail Dmitrievich     52              38              23              22
Tursunov Igor Erkindzhanovich   52              38              23              22
Kosarevsky Pyotr Ilyich         50              35              20              15
Kazantsev Danil Antonovich      48              36              26              22
Ozmanov David Tengizovich       48              30              19              18
Maximenko Artem Sergeyevich     45              32              24              20
---------------------------------------
```

Сортировка по 2 столбцу

```
Player Cards for Team: Sokol
Player                          Matches         GoalsScored     GoalsConceded   Assists
Kozlovsky Nikita Andreevich     48              32              25              20
Gudkov Yan Mikhailovich         45              28              18              10
Zabelin Pavel Vitalyevich       45              28              18              10
Kamyshov Ilya Vladislavovich    45              28              18              10
Azarov Vladimir Alexeevich      47              25              20              15
Molodtsov Artem Igorevich       42              22              22              13
Ezhkov Danila Andreevich        42              20              19              14
Ivakhnov Matvey Andreevich      37              18              16              12
---------------------------------------
```

Сортировка по 3 столбцу

```
Player Cards for Team: Spartak
Player                          Matches        GoalsScored    GoalsConceded  Assists
Fedorov Artem Anatolyevich      60             12             30             20
Sasin Dmitry Andreevich         42             30             22             17
Maximenko Artem Sergeyevich     45             25             20             18
Maltsev Mikhail Dmitrievich     45             25             20             18
Kazantsev Danil Antonovich      45             25             20             18
Yugaldin Dmitry Ilyich          45             25             20             18
Kamyshov Ilya Vladislavovich    40             24             18             15
Weber Dmitry Grigorievich       35             18             15             10
Tolstopyatov Nikolay Valeryevich 29            8              15             6
Mankov Nikita Sergeyevich       35             20             12             8
Dudiev Aslan Muratovich         25             5              10             7
---------------------------------------
```

Сортировка по 4 столбцу

```
---------------------------------------
Player Cards for Team: Zenit
Player                          Matches        GoalsScored    GoalsConceded  Assists
Karaev Maxim Irizalievich       55             30             35             20
Kazantsev Danil Antonovich      50             22             32             18
Maximenko Artem Sergeyevich     50             22             32             18
Maltsev Mikhail Dmitrievich     50             22             32             18
Tursunov Igor Erkindzhanovich   50             22             32             18
Morozov Vladislav Markovich     47             25             27             14
Batyrev Amir Renatovich         43             18             29             12
Azarov Vladimir Alexeevich      43             18             29             12
Serikov Ilya Andreevich         43             18             29             12
Pliev Zaurbek Igorevich         38             15             30             9
Gudkov Yan Mikhailovich         15             8              7              5
Abazov Astemir Valeryevich      40             0              35             2
---------------------------------------
```

Файл лога действий

```
(1) сформирован список игроков, сгруппированный по позиции и возрастной категории
(2) сформирован список игроков и кандидатов, которые играли ранее в одних командах
(3.1) сформированы учетные карточки на каждого игрока, упорядоченные по общему числу игр за все команды
(3.5) возвращение к общему меню
(4) сформированы учетные карточки на каждого игрока команды, упорядоченные по общему числу игр, забитых и пропущенных голов, голевых передач за команду
работа программы завершена
```

Файл лога памяти

```
Head NULL
---------------------------------------
 | /|\
\|/ |
---------------------------------------
id:      1
name:    Head -> Kosar->evsky-> Pyot->r Ily->ich -> NULL
year:    1999
city:    Head -> Dusha->nbe -> NULL
position:        Head -> Goalk->eeper -> NULL
status:  Head -> Team ->membe->r -> NULL
teamName:        Head -> Istik->lol -> NULL
commonPlayedMatches:     50
commonGoalsScored:       35
commonGoalsConceded:     20
commonAssists:   15
---------------------------------------
 | /|\
\|/ |
---------------------------------------
id:      2
name:    Head -> Kraik->ov Ti->mur V->iktor->ovich -> NULL
year:    2001
city:    Head -> Minsk -> NULL
position:        Head -> Goalk->eeper -> NULL
status:  Head -> Team ->membe->r -> NULL
teamName:        Head -> Torpe->do -> NULL
commonPlayedMatches:     30
commonGoalsScored:       2
commonGoalsConceded:     25
commonAssists:   5
---------------------------------------
 | /|\
\|/ |
---------------------------------------
id:      32
name:    Head -> Tursu->nov I->gor E->rkind->zhano->vich -> NULL
year:    1983
city:    Head -> Minsk -> NULL
position:        Head -> Forwa->rd -> NULL
status:  Head -> Candi->date -> NULL
teamName:        Head -> Torpe->do -> NULL Head -> Istik->lol -> NULL Head -> Zenit -> NULL
commonPlayedMatches:     142
commonGoalsScored:       75
commonGoalsConceded:     85
commonAssists:   55
---------------------------------------
 | /|\
\|/ |
---------------------------------------
id:      33
name:    Head -> Yugal->din D->mitry-> Ilyi->ch -> NULL
year:    1998
city:    Head -> Mosco->w -> NULL
position:        Head -> Midfi->elder -> NULL
status:  Head -> Candi->date -> NULL
teamName:        Head -> Spart->ak -> NULL
commonPlayedMatches:     45
commonGoalsScored:       25
commonGoalsConceded:     20
commonAssists:   18
---------------------------------------
 | /|\
\|/ |
---------------------------------------
NULL Tail
```

Вывод

В процессе написания курсовой работы были закреплены навыки, полученные в течении семестра, получены навыки откладки сложных иерархических программ, в результате была реализована программа обеспечивающая требуемый функционал.