```
/*main.cpp

#include "Start.h"
using namespace std;

int main()
{
     SetConsoleCP(1251);
     SetConsoleOutputCP(1251);
     setlocale(LC_ALL, "RUS");
     system("color 2");
     Fut_Team obj;
     obj.Start();
     return 0;
}

***************************************************

/*Start.h

#pragma once

#include "Defender.h"

#include "StackOtmena.h"

#include "Forward.h"

#include "Fis_Trener.h"

#include "Vera_Trener.h"

#include "String.h"

#include "Shablon.cpp"

#include "Iskluch.h"

#include "TXTFILE.h"

#include <list>

#include <exception>

#include <vector>

#include "Algoritm.h"

#define n1 "Chlen_komandi"

#define n2 "Footballist"

#define n3 "Defender"

#define n4 "Forward"

#define n5 "Trener"

#define n6 "Fis_Trener"

#define n7 "Vera_Trener"
```

```
class Fut_Team
{
public:
     int Start();
     string print_table(int _b);
     template<typename TYPE>
     void view(TYPE _obj, int b, list<TYPE> _c);
     template<typename TYPE>
     void stl(list<TYPE> _c, int _b);
};


string Fut_Team:: print_table(int _b)
{

     rewind(stdin);
     if (_b == 1)
     {
          cout << "---------------------------------------------
-------" << endl;
          cout << setiosflags(ios::left)
               << setw(20) << " NAME"
               << setw(20) << " SURNAME"
               << setw(10) << "YEAR"
               << endl;
          cout << "---------------------------------------------
-------";
          cout << endl;
          return n1;
     }
     if (_b == 2)
     {
          cout << "---------------------------------------------
----------------" << endl;
          cout << setiosflags(ios::left)
               << setw(20) << " NAME"
               << setw(20) << " SURNAME"
               << setw(10) << "YEAR"
               << setw(10) << "NOMER"
               << endl;
          cout << "---------------------------------------------
----------------";
          cout << endl;
          return n2;
     }
     if (_b == 3)
     {
          cout << "---------------------------------------------
-------------------------" << endl;
          cout << setiosflags(ios::left)
               << setw(20) << " NAME"
               << setw(20) << " SURNAME"
               << setw(10) << "YEAR"
               << setw(10) << "NOMER"
               << setw(10) << "Yellow card"
               << endl;
          cout << "---------------------------------------------
-------------------------";
          cout << endl;
          return n3;
     }
     if (_b == 4)
     {
          cout << "---------------------------------------------
-----------------------------------" << endl;
          cout << setiosflags(ios::left)
```

```
               << setw(20) << " NAME"
               << setw(20) << " SURNAME"
               << setw(10) << "YEAR"
               << setw(10) << "NOMER"
               << setw(10) << "GOALS"
               << setw(10) << "ASSISTS"
               << endl;
          cout << "---------------------------------------------
-----------------------------------";
          cout << endl;
          return n4;
     }
     if (_b == 5)
     {
          cout << "---------------------------------------------
----------------" << endl;
          cout << setiosflags(ios::left)
               << setw(20) << " NAME"
               << setw(20) << " SURNAME"
               << setw(10) << "YEAR"
               << setw(10) << "Staj"
               << endl;
          cout << "---------------------------------------------
----------------";
          cout << endl;
          return n5;
     }
     if (_b == 6)
     {
          cout << "---------------------------------------------
-----------------------------------" << endl;
          cout << setiosflags(ios::left)
               << setw(20) << " NAME"
               << setw(20) << " SURNAME"
               << setw(10) << "YEAR"
               << setw(10) << "Staj"
               << setw(20) << " Vid trenirovki"
               << endl;
          cout << "---------------------------------------------
-----------------------------------";
          cout << endl;
          return n6;
     }
     if (_b == 7)
     {
          cout << "---------------------------------------------
-------------------------" << endl;
          cout << setiosflags(ios::left)
               << setw(20) << " NAME"
               << setw(20) << " SURNAME"
               << setw(10) << "YEAR"
               << setw(10) << "Staj"
               << setw(10) << "Vremia trenirovki"
               << endl;
          cout << "---------------------------------------------
-------------------------";
          cout << endl;
          return n7;
     }
}
```

```
template<typename TYPE>
void Fut_Team:: view(TYPE _obj, int b, list<TYPE> _c)
{
	cout << "Количество?" << endl;
	int z;
	z = enter_int();
	linklist <TYPE> a;
	Stack<TYPE> otm;
	TYPE* tmp1;
	TYPE s, l;
	tmp1 = new TYPE[z];
	for (int i = 0; i < z; i++)
	{
		cout << "---------" << endl;
		rewind(stdin);
		cin >> tmp1[i];
		a.sozd(tmp1[i]);
	}
	string k;
	k = print_table(b);
	a.pop_data();
	char w;
	int cc;
	char A[80];
	TYPE ss;
	Algorithm<TYPE> alg;
	while (1)
	{
		cout << endl;
		cout << "Выберите тип: 1-Добавить в шаблон(в любую 
позицию)\n";
		cout << "              2-Вывести шаблон\n";
```

```
          cout << "               3-Поиск по названию\n";
          cout << "               4-Поиск по индексу\n";
          cout << "               5-Удаление объекта\n";
          cout << "               6-Отмена удаления\n";
          cout << "               7-Отмена добавления\n";
          cout << "               8-Прочитать из текстового
файла\n";
          cout << "               9-Записать в текстовый файл\n";
          cout << "               10-Очистить текстовый файл\n";
          cout << "               11-показать в обратном
порядке(с итератором)\n";
          cout << "               12-показать(с итератором)\n";
          cout << "               13-поиск по названию(с
итератором)\n";
          cout << "               14-сортировка\n";
          cout << "               15-STL\n";
          cout << "               16-выход\n";
          rewind(stdin);
          cc = enter_int();
          switch (cc)
          {
          case 1:
          {
               system("cls");
               k = print_table(b);
               a.pop_data();
               cout << endl;
               cin >> s;
               cout << endl << "Введите позицию для добавления"
<< endl;
               cc = enter_int();
               a.add_element(s, cc);
               otm.add_data(s);
```

```
        a.pop_data();
        break;
    }
    case 2:
    {
        system("cls");
        a.pop_data();
        break;
    }
    case 3:
    {
        system("cls");
        a.pop_data();
        cin >> ss;
        alg.search2(a.Begin(), ss);
        break;
    }
    case 4:
    {
        system("cls");
        a.pop_data();
        cout << "Введи индекс" << endl;
        cc = enter_int();
        if (a.getrazmer() >= cc)
            alg.search1(a.Begin(), cc);
        break;
    }
    case 5:
    {
        system("cls");
        a.pop_data();
```

```
                cout << endl << "Введите что хотите удалить" <<
endl;
                cin >> ss;
                a.udalenie(ss);
                otm.add_data(ss);
                a.pop_data();
                break;
           }
           case 6:
           {
                s = otm.del();
                a.add_data(s);
                cout << "Отмена удаления прошла успешно." <<
endl;
                break;
           }
           case 7:
           {
                s = otm.del();
                a.udalenie(s);
                cout << "Отмена добавления прошла успешно." <<
endl;
                break;
           }
           case 8:
           {
                system("cls");
                TXTFILE _b;
                linklist <TYPE> c;
                TYPE* tmp2;
                string q;
                q = k + ".txt";
                int count;
                count = _b.checkCount(q);
                tmp2 = new TYPE[count];
                for (int i = 0; i < count; i++)
                {
                     _b.fromFile(tmp2[i], q, i);
                     c.add_data(tmp2[i]);
                }
                cout << "Данные успешно считаны из файла." <<
endl;
                print_table(b);
                c.pop_data();
                break;
           }
           case 9:
           {
                system("cls");
                string q;
                q = k + ".txt";
                vector<TYPE> ww;
                ww = a.get();
                TXTFILE b;
                for (int i = 0; i < a.getrazmer(); i++)
                {
                     b.toFile(ww[i], q);
                }
                cout << "Данные успешно записаны в файл." <<
endl;
                break;
           }
           case 10:
           {
                system("cls");
                string q;
```

```
          q = k + ".txt";
          TXTFILE b;
          b.clear(q);
          cout << "Файл очищен." << endl;
          break;
     }
     case 11:
     {
          system("cls");
          a.pop_data();
          cout << endl;
          a.printback();
          break;
     }
     case 12:
     {
          system("cls");
          a.pop_data();
          cout << endl;
          a.printIterator();
          break;
     }
     case 13:
     {
          system("cls");
          a.pop_data();
          cin >> ss;
          alg.search2Iterator(a.Begin(), a.End(), ss);
          break;
     }
     case 14:
     {
```

```
				system("cls");
				a.pop_data();
				alg.sort(a);
				cout << endl << "После сортировки" << endl;
				a.pop_data();
				break;
			}
			case 15:
			{
				stl(_c, b);
				break;
			}
			case 16:
			{
				return;
			}
			}
			cout << endl;
			system("pause");
		}
}



template<typename TYPE>
void Fut_Team::stl(list<TYPE> _c, int _b)
{
	system("cls");
	list<TYPE> a;
	char w;
	TYPE d;
	while (1)
```

```
     {
          cout << endl << "Выбери операцию" << endl;
          cout << "1-Добавить в конец list" << endl << "2-
Добавить в начало list" << endl << "3-Показать list" << endl <<
"4-Удалить элемент" << endl
               << "5-Очистить list" << endl << "6-Реверс
элементов" << endl << "7-Узнать размер list"
               << endl << "8-Убрать совпадающие элементы list"
<< endl << "9-Вставить элемент в list" << endl << "10-Выход" <<
endl;
          w = enter_int();
          switch (w)
          {
          case 1:
          {
               system("cls");
               cin >> d;
               a.push_back(d);
               system("cls");
               cout << "Объект успешно добавлен" << endl;
               break;
          }
          case 2:
          {
               system("cls");
               cin >> d;
               a.push_front(d);
               system("cls");
               cout << "Объект успешно добавлен" << endl;
               break;
          }
          case 3:
          {
               system("cls");
```

```
                print_table(_b);
                list<TYPE>::template iterator at;
                for (at = a.begin(); at != a.end(); at++)
                {
                     cout << (*at);
                     cout << endl;
                }
                break;
           }
           case 4:
           {
                system("cls");
                print_table(_b);
                list<TYPE>::template iterator at;
                for (at = a.begin(); at != a.end(); at++)
                {
                     cout << (*at);
                     cout << endl;
                }
                cout << "Какой элемент удалить?(Название)" <<
endl;
                string t;
                cin >> t;
                for (at = a.begin(); at != a.end(); at++)
                {
                     if ((at->GetName()) == t)
                     {
                          a.erase(at);
                          system("cls");
                          cout << "Объект удалён" << endl;
                          break;
                     }
```

```
     }
     break;
}
case 5:
{
     system("cls");
     a.clear();
     cout << "list очищен" << endl;
     break;
}
case 6:
{
     system("cls");
     a.reverse();
     cout << "Реверс list" << endl;
     break;
}
case 7:
{
     system("cls");
     int i;
     i = a.size();
     cout << "Размер list = " << i << endl;
     break;
}
case 8:
{
     system("cls");
     a.unique();
     cout << "Совпадающие элементы удалены" << endl;
     break;
}
```

```
          case 9:
          {
               system("cls");
               cin >> d;
               cout << "Введи позицию для вставки элемента" <<
endl;
               int i, kl = 0;
               cin >> i;
               list<TYPE>::template iterator at;
               for (at = a.begin(); at != a.end(); at++)
               {
                    kl++;
                    if (kl == i)
                    {
                         a.insert(at, d);
                         break;
                    }
               }
               cout << "Элемент добавлен" << endl;
               break;
          }
          case 10:
          {
               return;
          }
          }
     }
}


int Fut_Team::Start()
{
     int zz;

     cout << "\n\t\t\t*********** СВЕДЕНИЯ ОБ ИГРОКАХ ФУТБОЛЬНОЙ
КОМАНДЫ ***********\n\n\n\n";

     while (1)

     {

          cout << "Выберите команду: 1-Добавить и показать
информацию Chlen_komandi\n";

          cout << "                  2-Добавить и показать
информацию Footballist\n";

          cout << "                  3-Добавить и показать
информацию Defender\n";

          cout << "                  4-Добавить и показать
информацию Forward\n";

          cout << "                  5-Добавить и показать
информацию Trener\n";

          cout << "                  6-Добавить и показать
информацию Fis_Trener\n";

          cout << "                  7-Добавить и показать
информацию Vera_Trener\n";

          cout << "                  0-Выйти\n";

          rewind(stdin);

          zz = enter_int();

          switch (zz)

          {

          case 1:

          {

               list<Chlen_komandi> c;

               Chlen_komandi t1;

               int b = 1;

               view(t1, b, c);

               break;

          }

          case 2:

          {

               list<Footballist> c;
               Footballist m1;
               int b = 2;
               view(m1, b, c);
               break;
          }
          case 3:
          {
               list<Defender> c;
               Defender l1;
               int b = 3;
               view(l1, b, c);
               break;
          }
          case 4:
          {
               list<Forward> c;
               Forward me1;
               int b = 4;
               view(me1, b, c);
               break;
          }
          case 5:
          {
               list<Trener> c;
               Trener a1;
               int b = 5;
               view(a1, b, c);
               break;
          }
          case 6:
          {
               list<Fis_Trener> c;
```

```
				Fis_Trener ma1;
				int b = 6;
				view(ma1, b, c);
				break;
			}
			case 7:
			{
				list<Vera_Trener> c;
				Vera_Trener ma1;
				int b = 7;
				view(ma1, b, c);
				break;
			}
			case 0: return 0;
			default: cout << "Ошибка, повторите \n";
			}
			cout << endl;
			system("pause");
		}
		system("pause");
		return 0;
	}
	***********************************************
	/*Chlen_comandi.h
	#pragma once
	#include "String.h"
	using namespace std;

	class Chlen_komandi
	{
	protected:
		String name;
		String surname;
		int year;
	public:
		Chlen_komandi()
		{
```

```
		name = "";
		surname = "";
		year = 0;
	}
	Chlen_komandi(String str,String str1, int value = 0)
	{
		name = str;
		surname = str1;
		year = value;
	}
	friend istream& operator>>(istream& in, Chlen_komandi& ob)
	{
		cout << "Введите имя члена команды";
		char* pr_str(istream & in);
		ob.name = pr_str(in);
		cout << "Введите фамилию члена команды";
		char* pr_str(istream & in);
		ob.surname = pr_str(in);
		cout << "Введите возраст игрока:";
		int enter_int(istream & in);
		ob.year=enter_int(in);
		return in;
	}
	friend ostream& operator<<(ostream& on, Chlen_komandi& ob)
	{
		on << setw(20) << ob.name << setw(20) << ob.surname <<
setw(10) << ob.year;
		return on;
	}

	friend void operator <<= (std::ostream& stream,
Chlen_komandi& tmp)
	{
		stream << tmp.name << "|" << tmp.surname << "|" <<
tmp.year << "|";
	}

	friend void operator >>= (std::istream& stream,
Chlen_komandi& tmp)
	{
		string s, s1;
		if (getline(stream, s))
		{
			stringstream ss;
			ss << s;
			getline(ss, s1, '|');
			tmp.SetName(s1.c_str());
			getline(ss, s1, '|');
			tmp.SetSurname(s1.c_str());
			getline(ss, s1, '|');
			tmp.SetYear(atoi(s1.c_str()));
		}
	}
	friend void operator <= (std::ostream& os, Chlen_komandi& 
tmp);
	friend void operator >= (std::istream& is, Chlen_komandi& 
tmp);

	bool operator==(Chlen_komandi& obj)
	{
		bool m;
		if (name == "" && year != 0 && surname == "")
		{
			return m = (year == obj.year);
		}
		if (name != "" && year == 0 && surname == "")
		{
			return m = (name == obj.name);
		}
		if (name == "" && year == 0 && surname != "")
		{
			return m = (surname == obj.surname);
		}
		if (name != "" && year != 0 && surname == "")
		{
			return m = ((year == obj.year)||(name == 
obj.name));
		}
		if (name == "" && year != 0 && surname != "")
		{
			return m = ((year == obj.year) || (surname == 
obj.surname));
		}
		if (name != "" && year == 0 && surname != "")
		{
			return m = ((name == obj.name) || (surname == 
obj.surname));
		}
		if (name != "" && year != 0 && surname != "")
		{
			return m = ((year == obj.year) || (surname == 
obj.surname)|| (name == obj.name));
		}
	}

	void operator = (const Chlen_komandi& tmp);
	bool operator > (const Chlen_komandi& tmp);
	//Методы доступа к полям класса
	String GetName();
	void SetName(const char* str);
	void SetName(String str);
	String GetSurname();
	void SetSurname(const char* str);
	void SetSurname(String str);
	int GetYear();
	void SetYear(int value1);
```

```
     ~Chlen_komandi() {};
};

/*Chlen_komandi.cpp

#include "Chlen_komandi.h"

String Chlen_komandi:: GetName()
{
     return name;
};
void Chlen_komandi:: SetName(const char* str)
{
     name = str;
}
void Chlen_komandi::SetName(String str)
{
     name = str;
}
String Chlen_komandi::GetSurname()
{
     return surname;
};
void Chlen_komandi::SetSurname(const char* str)
{
     surname = str;
}
void Chlen_komandi::SetSurname(String str)
{
     surname = str;
}
int Chlen_komandi::GetYear()
{
     return year;
};
void Chlen_komandi::SetYear(int value1)
{
     year = value1;
}

void operator <= (std::ostream& os, Chlen_komandi& tmp)
{
     os.write(reinterpret_cast<const char*>(&tmp),
sizeof(Chlen_komandi));
     os.write(reinterpret_cast<const char*>(&tmp.year),
sizeof(int));
}

void operator >= (std::istream& is, Chlen_komandi& tmp)
{
     int n;
     is.read(reinterpret_cast<char*>(&tmp),
sizeof(Chlen_komandi));
```

```
	tmp.SetName(tmp.name);
	is.read(reinterpret_cast<char*>(&tmp),
sizeof(Chlen_komandi));
	tmp.SetSurname(tmp.surname);
	is.read(reinterpret_cast<char*>(&n), sizeof(int));
	tmp.SetYear(n);
}



void Chlen_komandi::operator = (const Chlen_komandi& tmp)
{
	/*this->god_proizvodstva = tmp.god_proizvodstva;*/
	if (this->name!="" && (this->name == tmp.name))
		return;
	name = tmp.name;
	surname = tmp.surname;
	year = tmp.year;
}


bool Chlen_komandi::operator > (const Chlen_komandi& tmp)
{
	if (this->name>tmp.name)
		return true;
	if (this->name<tmp.name)
		return false;
	if (this->name == tmp.name)
		return false;
}
```

```
***********************************************

/*Footballist.h

#pragma once
#include "Footballist.h"
class Forward :public Footballist
{
protected:
	int goals;
	int assists;
public:
	Forward() :Footballist()
	{
		goals = 0;
		assists = 0;
	}
	Forward(String ptr,int per = 0, int per1 = 0, int value1 =
0) :Footballist(ptr,value1)
	{
		goals = per;
		assists = per1;
	}
	friend istream& operator>>(istream& in, Forward& obj)
	{
		int flag;
		do
		{
			try
			{
				flag = 0;
				in >> dynamic_cast<Footballist&>(obj);
			}
			catch (const bad_cast& ob)
			{
				flag = 1;
				cout << "Error: " << ob.what();
			}
		} while (flag);
		cout << "Введите количество голов:";
		int enter_int(istream & in);
		obj.goals=enter_int(in);
		cout << "Введите количество голевых передач:";
		int enter_int(istream & in);
		obj.assists=enter_int(in);
		return in;
	}

	friend void operator <<= (std::ostream& stream, Forward&
tmp)
	{
		stream <<= dynamic_cast <Footballist&>(tmp);
		stream << tmp.goals << "|" << tmp.assists << "|";
	}

	friend void operator >>= (std::istream& stream, Forward&
tmp)
	{
		string s, s1;
		if (getline(stream, s))
		{
			stringstream ss;
			ss << s;
			getline(ss, s1, '|');
			tmp.SetName(s1.c_str());
			getline(ss, s1, '|');
			tmp.SetSurname(s1.c_str());
			getline(ss, s1, '|');
			tmp.SetYear(atoi(s1.c_str()));
			getline(ss, s1, '|');
			tmp.SetNumber(atoi(s1.c_str()));
			getline(ss, s1, '|');
			tmp.SetGoals(atoi(s1.c_str()));
			getline(ss, s1, '|');
			tmp.SetAssists(atoi(s1.c_str()));
```

```
          }
     }
     friend void operator <= (std::ostream& os, Forward& tmp);
     friend void operator >= (std::istream& is, Forward& tmp);
     friend ostream& operator << (ostream& out, Forward& obj)
     {
          out << setw(20) << obj.name
               << setw(20) << obj.surname
               << setw(10) << obj.year
               << setw(10) << obj.number
               << setw(10) << obj.goals
               << setw(10) << obj.assists;
               return out;
     }

     bool operator==(Forward& obj)
     {
          bool m;
          if (goals == 0 && assists != 0)
               return m = (assists != assists);
          if (goals != 0 && assists == 0)
               return m = (goals != goals);
          if (goals != 0 && assists != 0)
               return m = (goals != goals||assists!=assists);
     }
     int GetGoals();
     void SetGoals(int per);
     int GetAssists();
     void SetAssists(int per1);
     ~Forward() {};
};


/*Footballist.cpp

#include "Footballist.h"

int Footballist::GetNumber()
{
     return number;
};
void Footballist::SetNumber(int sss)
{
     number = sss;
}

void operator <= (std::ostream& os, Footballist& tmp)
{
     os <= dynamic_cast <Chlen_komandi&>(tmp);
     os.write(reinterpret_cast<const char*>(&tmp.number),
sizeof(int));
}
```

```
void operator >= (std::istream& is, Footballist& tmp)
{
     is >= dynamic_cast <Chlen_komandi&>(tmp);
     int n;
     is.read(reinterpret_cast<char*>(&n), sizeof(int));
     tmp.SetNumber(n);
}


**********************************************************

/*Trener.h

#pragma once
#include "Chlen_komandi.h"
class Trener :public Chlen_komandi
{
protected:
     int staj_work;
public:
     Trener() :Chlen_komandi()
     {
          staj_work = 0;
     }
     Trener(String _name,String name1,int _staj_work =0,int
_year = 0) :Chlen_komandi(_name,name1,_year)
     {
          this->staj_work = _staj_work;
     }
     friend ostream& operator<<(ostream& on, Trener& obj)
     {
          on << setw(20) << obj.name
               << setw(20) << obj.surname
               << setw(10) << obj.year
               << setw(10) << obj.staj_work;
          return on;
     }
     friend istream& operator>>(istream& in, Trener& obj)
     {
          int flag;
          do
          {
               try
               {
                    flag = 0;
                    in >> dynamic_cast<Chlen_komandi&>(obj);
               }
               catch (const bad_cast& ob)
               {
                    flag = 1;
                    cout << "Error: " << ob.what();
               }
          } while (flag);
          cout << "Введите стаж работы";
```

```
		int enter_int(istream & in);
		obj.staj_work=enter_int(in);
		return in;
	}
	friend void operator <<= (std::ostream& stream, Trener& 
tmp)
	{
		stream <<= dynamic_cast <Chlen_komandi&>(tmp);
		stream << tmp.staj_work << "|";
	}

	friend void operator >>= (std::istream& stream, Trener& 
tmp)
	{
		string s, s1;
		if (getline(stream, s))
		{
			stringstream ss;
			ss << s;
			getline(ss, s1, '|');
			tmp.SetName(s1.c_str());
			getline(ss, s1, '|');
			tmp.SetSurname(s1.c_str());
			getline(ss, s1, '|');
			tmp.SetYear(atoi(s1.c_str()));
			getline(ss, s1, '|');
			tmp.SetStaj_Work(atoi(s1.c_str()));
		}
	}
	friend void operator <= (std::ostream& os, Trener& tmp);
	friend void operator >= (std::istream& is, Trener& tmp);
	bool operator==(Trener& obj)
	{
		if (obj.staj_work == this->staj_work)
			return true;
		else
			return false;
	}
	int GetStaj_Work();
	void SetStaj_Work(int value1);
	~Trener() {}
};




/*Trener.cpp

#include "Trener.h"

int Trener::GetStaj_Work()
	{
	    return this->staj_work;
     }
void Trener::SetStaj_Work(int value1)
{
     this->staj_work = value1;
}

void operator <= (std::ostream& os, Trener& tmp)
{
     os <= dynamic_cast <Chlen_komandi&>(tmp);
     os.write(reinterpret_cast<const char*>(&tmp.staj_work),
sizeof(int));
}

void operator >= (std::istream& is, Trener& tmp)
{
     is >= dynamic_cast <Chlen_komandi&>(tmp);
     int n;
     is.read(reinterpret_cast<char*>(&n), sizeof(int));
     tmp.SetStaj_Work(n);
}
```

****
****

```
/*Forward.h

#pragma once
#include "Footballist.h"
class Forward :public Footballist
{
protected:
     int goals;
     int assists;
public:
     Forward() :Footballist()
     {
          goals = 0;
          assists = 0;
     }
     Forward(String ptr,int per = 0, int per1 = 0, int value1 =
0) :Footballist(ptr,value1)
     {
          goals = per;
          assists = per1;
     }
     friend istream& operator>>(istream& in, Forward& obj)
     {
          int flag;
          do
          {
               try
               {
                    flag = 0;
                    in >> dynamic_cast<Footballist&>(obj);
```

```
			}
			catch (const bad_cast& ob)
			{
				flag = 1;
				cout << "Error: " << ob.what();
			}
		} while (flag);
		cout << "Введите количество голов:";
		int enter_int(istream & in);
		obj.goals=enter_int(in);
		cout << "Введите количество голевых передач:";
		int enter_int(istream & in);
		obj.assists=enter_int(in);
		return in;
	}

	friend void operator <<= (std::ostream& stream, Forward& 
tmp)
	{
		stream <<= dynamic_cast <Footballist&>(tmp);
		stream << tmp.goals << "|" << tmp.assists << "|";
	}

	friend void operator >>= (std::istream& stream, Forward& 
tmp)
	{
		string s, s1;
		if (getline(stream, s))
		{
			stringstream ss;
			ss << s;
			getline(ss, s1, '|');
			tmp.SetName(s1.c_str());
			getline(ss, s1, '|');
			tmp.SetSurname(s1.c_str());
			getline(ss, s1, '|');
			tmp.SetYear(atoi(s1.c_str()));
			getline(ss, s1, '|');
			tmp.SetNumber(atoi(s1.c_str()));
			getline(ss, s1, '|');
			tmp.SetGoals(atoi(s1.c_str()));
			getline(ss, s1, '|');
			tmp.SetAssists(atoi(s1.c_str()));
		}
	}
	friend void operator <= (std::ostream& os, Forward& tmp);
	friend void operator >= (std::istream& is, Forward& tmp);
	friend ostream& operator << (ostream& out, Forward& obj)
	{
		out << setw(20) << obj.name
			<< setw(20) << obj.surname
			<< setw(10) << obj.year
			<< setw(10) << obj.number
```

```
                << setw(10) << obj.goals
                << setw(10) << obj.assists;
                return out;
     }

     bool operator==(Forward& obj)
     {
          bool m;
          if (goals == 0 && assists != 0)
               return m = (assists != assists);
          if (goals != 0 && assists == 0)
               return m = (goals != goals);
          if (goals != 0 && assists != 0)
               return m = (goals != goals||assists!=assists);
     }
     int GetGoals();
     void SetGoals(int per);
     int GetAssists();
     void SetAssists(int per1);
     ~Forward() {};
};


/*Forward.cpp

#include "Forward.h"

int Forward::GetGoals()
{
     return goals;
};
void Forward::SetGoals(int per)
{
     goals = per;
}
int Forward::GetAssists()
{
     return assists;
};
void Forward::SetAssists(int per1)
{
     assists = per1;
}


void operator <= (std::ostream& os, Forward& tmp)
{
     os <= dynamic_cast <Footballist&>(tmp);
     os.write(reinterpret_cast<const char*>(&tmp.goals),
sizeof(int));
     os.write(reinterpret_cast<const char*>(&tmp.assists),
sizeof(int));
}
```

```
void operator >= (std::istream& is, Forward& tmp)
{
     is >= dynamic_cast <Footballist&>(tmp);
     int n;
     is.read(reinterpret_cast<char*>(&n), sizeof(int));
     tmp.SetGoals(n);
     is.read(reinterpret_cast<char*>(&n), sizeof(int));
     tmp.SetAssists(n);
}
```

****

```
/*Defender.h

#pragma once
#include "Footballist.h"
class Defender :public Footballist
{
protected:
     int yellow_card;
public:
     Defender() :Footballist()
     {
          yellow_card = 0;
     }
     Defender(String ptr,int per = 0, int value1 = 0)
:Footballist(ptr,value1)
     {
          yellow_card = per;
     }
     friend istream& operator>>(istream& in, Defender& obj)
     {
          int flag;
          do
          {
               try
               {
                    flag = 0;
                    in >> dynamic_cast<Footballist&>(obj);
               }
               catch (const bad_cast& ob)
               {
                    flag = 1;
                    cout << "Error: " << ob.what();
               }
          } while (flag);
          cout << "Введите количество желтых карточек:";
          int enter_int(istream & in);
          obj.yellow_card=enter_int(in);
          return in;
     }
     friend ostream& operator<<(ostream& on, Defender& obj)
     {
```

```
		on << setw(20) << obj.name
			<< setw(20) << obj.surname
			<< setw(10) << obj.year
			<< setw(10) << obj.number
			<< setw(10) << obj.yellow_card;
		return on;
	}
	friend void operator <<= (std::ostream& stream, Defender&
tmp)
	{
		stream <<= dynamic_cast <Footballist&>(tmp);
		stream << tmp.yellow_card << "|";
	}

	friend void operator >>= (std::istream& stream, Defender&
tmp)
	{
		string s, s1;
		if (getline(stream, s))
		{
			stringstream ss;
			ss << s;
			getline(ss, s1, '|');
			tmp.SetName(s1.c_str());
			getline(ss, s1, '|');
			tmp.SetSurname(s1.c_str());
			getline(ss, s1, '|');
			tmp.SetYear(atoi(s1.c_str()));
			getline(ss, s1, '|');
			tmp.SetYear(atoi(s1.c_str()));
			getline(ss, s1, '|');
			tmp.SetYellow_Card(atoi(s1.c_str()));
		}
	}
	friend void operator <= (std::ostream& os, Defender& tmp);
	friend void operator >= (std::istream& is, Defender& tmp);
	bool operator==(Defender& obj)
	{
		if (obj.yellow_card == this->yellow_card)
			return true;
		else
			return false;
	}
	int GetYellow_Card();
	void SetYellow_Card(int per);
	~Defender() {};
};



/*Defender.cpp

#include "Defender.h"
```

```
int Defender::GetYellow_Card()
{
     return yellow_card;
};
void Defender::SetYellow_Card(int per)
{
     yellow_card = per;
}

void operator <= (std::ostream& os, Defender& tmp)
{
     os <= dynamic_cast <Footballist&>(tmp);
     os.write(reinterpret_cast<const char*>(&tmp.yellow_card),
sizeof(int));
}

void operator >= (std::istream& is, Defender& tmp)
{
     is >= dynamic_cast <Footballist&>(tmp);
     int n;
     is.read(reinterpret_cast<char*>(&n), sizeof(int));
     tmp.SetYellow_Card(n);
}
****************************************************************
*

/*Fis_Trener

#pragma once
#include "Trener.h"


class Fis_Trener:public Trener
{
protected:
     String vid_treni;
public:
     Fis_Trener():Trener()
     {
          vid_treni = "";
     }
     Fis_Trener(String _vid_treni,String ptr, int per = 0) :
          Trener(ptr,per)
     {
          vid_treni = _vid_treni;
     }
     friend ostream& operator<<(ostream& on, Fis_Trener& obj)
     {
         on << setw(20) << obj.name
                << setw(20) << obj.surname
          << setw(10) << obj.year
          << setw(10) << obj.staj_work
```

```
                << setw(20) << obj.vid_treni;
               return on;
           }
           friend istream& operator>>(istream& in, Fis_Trener& obj)
           {
                int flag;
                do
                {
                     try
                     {
                          flag = 0;
                          in >> dynamic_cast<Trener&>(obj);
                     }
                     catch (const bad_cast& ob)
                     {
                          flag = 1;
                          cout << "Error: " << ob.what();
                     }
                } while (flag);
                cout << "Введите вид тренировки";
                char* pr_str(istream & in);
                obj.vid_treni = pr_str(in);
                return in;
           }

           friend void operator >>= (std::istream& stream, Fis_Trener&
      tmp)
           {
                string s, s1;
                if (getline(stream, s))
                {
                     stringstream ss;
                     ss << s;
                     getline(ss, s1, '|');
                     tmp.SetName(s1.c_str());
                     getline(ss, s1, '|');
                     tmp.SetSurname(s1.c_str());
                     getline(ss, s1, '|');
                     tmp.SetYear(atoi(s1.c_str()));
                     getline(ss, s1, '|');
                     tmp.SetStaj_Work(atoi(s1.c_str()));
                     getline(ss, s1, '|');
                     tmp.SetVid_Treni(s1.c_str());
                }
           }
           friend void operator <= (std::ostream& os, Fis_Trener&
      tmp);
           friend void operator >= (std::istream& is, Fis_Trener&
      tmp);

           friend void operator <<= (std::ostream& stream, Fis_Trener&
      tmp)
           {
```

```
          stream <<= dynamic_cast <Trener&>(tmp);
          stream << tmp.vid_treni << "|";
     }
     bool operator==(Fis_Trener& obj)
     {
          if (obj.vid_treni == this->vid_treni)
               return true;
          else
               return false;
     }
     String GetVid_Treni();
     void SetVid_Treni(const char* str);
     void SetVid_Treni(String str);
     ~Fis_Trener() {}
};

/*Fis_Trener.cpp
#include "Fis_Trener.h"

String Fis_Trener::GetVid_Treni()
{
     return this->vid_treni;
}
void Fis_Trener::SetVid_Treni(const char* str)
{
     vid_treni = str;
}

void Fis_Trener::SetVid_Treni(String str)
{
     vid_treni = str;
}


void operator <= (std::ostream& os, Fis_Trener& tmp)
{
     os <= dynamic_cast <Trener&>(tmp);
     os.write(reinterpret_cast<const char*>(&tmp),
sizeof(Fis_Trener));
}

void operator >= (std::istream& is, Fis_Trener& tmp)
{
     is >= dynamic_cast <Trener&>(tmp);
     int n;
     is.read(reinterpret_cast<char*>(&tmp), sizeof(Fis_Trener));
     tmp.SetVid_Treni(tmp.vid_treni);
}
*****************************************************************
********

/*Vera_Trener.h
#pragma once
```

```
#include "Trener.h"

class Vera_Trener:public Trener
{
protected:
     int vremia_trener;
public:
     Vera_Trener():Trener()
     {
          vremia_trener = 0;
     }
     Vera_Trener(String ptr,int _vremia_trener=0,int per=0)
:Trener(ptr,per)
     {
          vremia_trener = _vremia_trener;
     }
     friend ostream& operator<<(ostream& on, Vera_Trener& obj)
     {
          on << setw(20) << obj.name
               << setw(20) << obj.surname
               << setw(10) << obj.year
               << setw(10) << obj.staj_work
               << setw(10) << obj.vremia_trener;
          return on;
     }
     friend istream& operator>>(istream& in, Vera_Trener& obj)
     {
         int flag;
         do
         {
          try
          {
               flag = 0;
               in >> dynamic_cast<Trener&>(obj);
          }
          catch (const bad_cast& ob)
          {
               flag = 1;
               cout << "Error: " << ob.what();
          }
         } while (flag);
         cout << "Введите время тренировки вратарей(в минутах)";
          int enter_int(istream & in);
         obj.vremia_trener=enter_int(in);
         return in;
     }

     friend void operator <<= (std::ostream& stream,
Vera_Trener& tmp)
     {
          stream <<= dynamic_cast <Trener&>(tmp);
          stream << tmp.vremia_trener << "|";
     }
```

```
	friend void operator >>= (std::istream& stream, 
Vera_Trener& tmp)
	{
		string s, s1;
		if (getline(stream, s))
		{
			stringstream ss;
			ss << s;
			getline(ss, s1, '|');
			tmp.SetName(s1.c_str());
			getline(ss, s1, '|');
			tmp.SetSurname(s1.c_str());
			getline(ss, s1, '|');
			tmp.SetYear(atoi(s1.c_str()));
			getline(ss, s1, '|');
			tmp.SetStaj_Work(atoi(s1.c_str()));
			getline(ss, s1, '|');
			tmp.SetVremia_Trener(atoi(s1.c_str()));
		}
	}
	friend void operator <= (std::ostream& os, Vera_Trener& 
tmp);
	friend void operator >= (std::istream& is, Vera_Trener& 
tmp);

	bool operator==(Vera_Trener& obj)
	{
		if (obj.vremia_trener == this->vremia_trener)
			return true;
		else
			return false;
	}
	int GetVremia_Trener();
	void SetVremia_Trener(int value);
	~Vera_Trener(){}
};

/*Vera_Trener.cpp
#include "Vera_Trener.h"
int Vera_Trener::GetVremia_Trener()
{
	return this->vremia_trener;
}
void Vera_Trener::SetVremia_Trener(int per)
{
	vremia_trener = per;
}


void operator <= (std::ostream& os, Vera_Trener& tmp)
{
	os <= dynamic_cast <Trener&>(tmp);
```

```
     os.write(reinterpret_cast<const char*>(&tmp.vremia_trener),
sizeof(int));
}

void operator >= (std::istream& is, Vera_Trener& tmp)
{
     is >= dynamic_cast <Trener&>(tmp);
     int n;
     is.read(reinterpret_cast<char*>(&n), sizeof(int));
     tmp.SetVremia_Trener(n);
}
****************************************************************
*****
/*Algoritm.h
#pragma once
#include "Shablon.h"

template <class TYPE>
class Algorithm
{
public:
     Algorithm();
     ~Algorithm();
     TYPE search1(Node<TYPE>* beg, int c);
     linklist<TYPE>& search2(Node<TYPE>* beg, TYPE _obj);
     linklist<TYPE>& search2Iterator(Node<TYPE>* beg,
Node<TYPE>* end, TYPE _obj);
     void sort(linklist<TYPE>& _a);
};

template <class TYPE>
Algorithm<TYPE>::Algorithm()
{
}

template <class TYPE>
Algorithm<TYPE>::~Algorithm()
{
}

template<typename TYPE>
TYPE Algorithm<TYPE>::search1(Node<TYPE>* beg, int c)
{
     TYPE temp;
     Node <TYPE>* rab = beg;
     while (c > 1)
     {
          rab = rab->next;
          c--;
     }
     cout << rab->data;
     temp = rab->data;
     return temp;
```

```
}

template<class TYPE>
linklist<TYPE>& Algorithm<TYPE>::search2(Node<TYPE>* beg, TYPE
_obj)
{
     Node <TYPE>* rab = beg;
     linklist <TYPE> temp;
     int fl = 0;
     while (rab != NULL)
     {
          if (rab->data == _obj)
          {
               cout << rab->data;
               fl = 1;
               temp.add_element(rab->data, 1);
          }
          rab = rab->next;
     }
     if (fl == 0)
     {
          cout << "Ошибка" << endl;
     }
     cout << endl;
     return temp;
}

template<class TYPE>
linklist<TYPE>& Algorithm<TYPE>::search2Iterator(Node<TYPE>*
beg, Node<TYPE>* end, TYPE _obj)
{
     Node <TYPE>* rab = beg;
     linklist <TYPE> temp;
     int fl = 0;
     Iterator<TYPE> it;
     for (it = beg; it != end->next; ++it)
     {
          if (*it == _obj)
          {
               cout << *it;
               fl = 1;
               temp.add_element((*it), 1);
          }
     }
     if (fl == 0)
     {
          cout << "Ошибка" << endl;
     }
     return temp;
}

template<class TYPE>
void Algorithm<TYPE>::sort(linklist<TYPE>& _a)
```

```
{
     Node<TYPE>* tmp = new Node<TYPE>;
     Iterator<TYPE> it_1 = _a.Begin();
     Iterator<TYPE> it_2 = _a.Begin()->next;
     while (it_1 != NULL)
     {
          it_2 = it_1;
          ++it_2;
          while (it_2 != NULL)
          {
               if ((*it_1) > (*it_2))
               {
                    tmp->data = *it_1;
                    *it_1 = *it_2;
                    *it_2 = tmp->data;
               }
               ++it_2;
          }
          ++it_1;
     }
}
****************************************************************
/*Exp_vvod.h
#pragma once
#include <iostream>
using namespace std;

class Exp_vvod
{
     int number;
     char ch[80];
public:
     Exp_vvod(const Exp_vvod& temp)
     {
          number = temp.number;
          strcpy_s(ch, strlen(temp.ch) + 1, temp.ch);
     }
     Exp_vvod(int _number, const char* str)
     {
          strcpy_s(ch, strlen(str) + 1, str);
          number = _number;
     }
     void show()
     {
          for (int i = 0; ch[i]; i++)
               cout << ch[i];
          cout << endl;
     }
     ~Exp_vvod()
     {

     }
};
```

```
****************************************************************
****
/*Iskluch.h
#pragma once
#include "Exp_vvod.h"
#include <iomanip>
#include <conio.h>
#include <process.h>
#include <windows.h>
#include <sstream>
#include <string>

char* pr_str(istream& in)
{
     int fe;
     char buf[80];
     char* str= new char[80];
     do
     {
          rewind(stdin);
          try
          {
               fe = 0;
               in.getline(buf, 80);
               strcpy_s(str, strlen(buf) + 1, buf);
               for (int i = 0; str[i]; i++)
               {
                    if (str[i] == ' ')
                         i++;
                    else if (str[i] < 'A' || str[i]>'z')
                         throw Exp_vvod(1, "Error: Написано не
на английском языке");
               }

          }
          catch (Exp_vvod ob)
          {
               fe = 1;
               ob.show();
               rewind(stdin);
          }
     } while (fe);
     return str;
}

int enter_int()
{
     int x;
     bool flag;
     do
     {
          rewind(stdin);
          try
          {
               rewind(stdin);
               flag = false;
               cin >> x;
               if (!cin.good() || cin.peek() != '\n'||x<0)
                    throw overflow_error("Введено не целое число 
или отрицательное");
          }
          catch (overflow_error ob)
          {
               cin.clear();
               rewind(stdin);
               flag = true;
               cout << "Error: " << ob.what() << endl;
          }
     } while (flag);
     return x;
}

int enter_int(istream& in)
{
     int x;
     bool flag;
     do
     {
          rewind(stdin);
          try
          {
               rewind(stdin);
               flag = false;
               in >> x;
               if (!in.good() || in.peek() != '\n' || x < 0)
                    throw overflow_error("Введено не целое число 
или отрицательное");
          }
          catch (overflow_error ob)
          {
               in.clear();
               rewind(stdin);
               flag = true;
               cout << "Error: " << ob.what() << endl;
          }
     } while (flag);
     return x;
}
****************************************************************
*********
/*Shablon.h
#pragma once
#include <iostream>
#include <process.h>
#include <string>
#include <conio.h>
```

```
#include <iomanip>
#include <algorithm>
#include <Windows.h>
#include <vector>
using namespace std;

template<class TYPE>
struct Node
{
	TYPE data;//данный
	Node* next;//указатель на след элемент
	Node* pred;//указатель на предыдущий элемент
};
template<class TYPE>
class linklist
{
private:
	Node<TYPE>* first;
	Node<TYPE>* tail;
	int size;
public:
	linklist()
	{
		first = nullptr;
		tail = nullptr;
		size = 0;
	}
	linklist(const linklist<TYPE>& obj)
	{
		first = obj.first;
		tail = obj.tail;
		size = obj.size;
	}
	template <typename TYPE>
	friend class Iterator;
	void printIterator();
	void printback();
	void sozd(TYPE note);
	TYPE del(TYPE value);
	void add_data(TYPE data);
	void pop_data();
	void udalenie(TYPE value);
	void ochistka();
	void show_size();
	void add_element(TYPE note, int poz);
	Node<TYPE>* Begin() { return first; }
	Node<TYPE>* End() { return tail; }
	vector<TYPE> get();
	int getrazmer();
	~linklist()
	{
		ochistka();
	}
```

```
};

/*Shablon.cpp
#include "Shablon.h"



template<typename TYPE>
void linklist<TYPE>::add_element(TYPE note, int poz)
{
     Node<TYPE>* rab = new Node <TYPE>;
     Node<TYPE>* tmp = new Node <TYPE>;
     tmp = first;
     if (size == 0)
     {
          rab->data = note;
          rab->pred = NULL;
          rab->next = NULL;
          first = tail = rab;
          size++;
          return;
     }
     if (poz == 1)
     {
          rab->data = note;
          rab->pred = NULL;
          rab->next = tmp;
          tmp->pred = rab;
          first = rab;
          size++;
          return;
     }
     if (poz == size + 1)
     {
          rab->data = note;
          tmp = tail;
          rab->pred = tmp;
          rab->next = NULL;
          tmp->next = rab;
          tail = rab;
          size++;
          return;
     }
     while (poz > 1)
     {
          tmp = tmp->next;
          poz--;
     }
     rab->data = note;
     rab->next = tmp;
     rab->pred = tmp->pred;
     tmp->pred->next = rab;
     tmp->pred = rab;
     size++;
}

template<class TYPE>
void linklist<TYPE>::show_size()
{
     cout << "Размер: " << size << endl;
}

template<class TYPE>
void linklist<TYPE>::add_data(TYPE data)
{
     ++size;
     Node<TYPE>* new_element = new Node<TYPE>;
     new_element->data = data;
     new_element->next = nullptr;
     new_element->pred = nullptr;
     if (!first || !tail)
     {
          first = tail = new_element;
     }
     else
     {
          new_element->next = first;
          new_element->pred = nullptr;
          first->pred = new_element;
          first = new_element;
     }
}


template<class TYPE>
void linklist<TYPE>::sozd(TYPE note)
{
     Node <TYPE>* rab = new Node <TYPE>;
     rab->next = NULL;
     rab->data = note;
     size++;
     if (first != NULL)
     {
          rab->pred = tail;
          tail->next = rab;
          tail = rab;
     }
     else
     {
          rab->pred = NULL;
          first = tail = rab;
     }
}


template<class TYPE>
```

```
void linklist<TYPE>::pop_data()
{
	/*cout << "Вывод листа" << endl;*/
	if (!first)
	{
		cout << "\nЛист пустой" << endl;
		return;
	}
	Node<TYPE>*n =first;
	while (n)
	{
		cout << endl << n->data;
		n = n->next;
	}
	cout << endl;
}

template<class TYPE>
void linklist<TYPE>::udalenie(TYPE value)
{
	Node<TYPE>*n = tail;
	if (!first || !tail)
	{
		cout << "Лист пустой";
		return;
	}
	if (n->pred == NULL && (n->data==value))
	{
		delete n;
		size--;
		tail = first = nullptr;
		return;
	}
	while (n && !(n->data==value))
	{
		n = n->pred;
	}
	if (n->data==value)
	{
		if (n == tail)
		{
			tail = tail->pred;
			tail->next = nullptr;
		}
		else if (n == first)
		{
			first = first->next;
			first->pred = nullptr;
		}
		else
		{
			n->next->pred = n->pred;
			n->pred->next = n->next;
```

```
          }
          delete n;
          size--;
     }
}



template<class TYPE>
TYPE linklist<TYPE>:: del(TYPE value)
{
     Node<TYPE>* n = tail;
     TYPE per;
     if (!first || !tail)
     {
          cout << "Лист пустой";
          return value;
     }
     if (n->pred == NULL && (n->data == value))
     {
          per = n->data;
          delete n;
          size--;
          tail = first = nullptr;
          return per;
     }
     while (n && !(n->data == value))
     {
          n = n->pred;
     }
     if (n->data == value)
     {
          if (n == tail)
          {
               tail = tail->pred;
               tail->next = nullptr;
          }
          else if (n == first)
          {
               first = first->next;
               first->pred = nullptr;
          }
          else
          {
               n->next->pred = n->pred;
               n->pred->next = n->next;
          }
          per = n->data;
          delete n;
          size--;
          return per;
     }
}
```

```
template<typename TYPE>
void linklist <TYPE>::printback()
{
     Iterator<TYPE> it;
     for (it = this->End(); it != this->Begin()->pred; --it)
     {
          cout << *it;
          cout << endl;
     }
}

template<typename TYPE>
void linklist <TYPE>::printIterator()
{
     Iterator<TYPE> it;
     for (it = this->Begin(); it != this->End()->next; ++it)
     {
          cout << *it;
          cout << endl;
     }
}

template<class TYPE>
vector<TYPE> linklist<TYPE>::get()
{
     vector<TYPE> ww;
     Iterator<TYPE> it;
     for (it = this->Begin(); it != this->End()->next; ++it)
     {
          ww.push_back(*it);
     }
     return ww;
}

template<class TYPE>
int  linklist<TYPE>::getrazmer()
{
     return this->size;
}

template <class TYPE>
void linklist<TYPE>::ochistka()
{
     while (first != nullptr)
     {
          Node<TYPE>* n = first->next;
          delete first;
          first = n;
     }
     tail = nullptr;
     size = 0;
}

template<typename TYPE>
class Iterator
{
private:
     Node<TYPE>* ptr;
public:
     Iterator()
     {
          ptr = NULL;
     }
     Iterator(Node<TYPE>* tmp)
     {
          ptr = tmp;
     }

     Iterator(const Iterator& tmp) : ptr(tmp.ptr) {}
     ~Iterator() { }
     Iterator& operator++()
     {
          if (ptr->next == NULL)
          {
               ptr = NULL;
               return *this;
          }
          ptr = ptr->next;
          return *this;
     }
     Iterator& operator--()
     {
          if (ptr->pred == NULL)
          {
               ptr = NULL;
               return *this;
          }
          ptr = ptr->pred;
          return *this;
     }

     TYPE& operator*()
     {
          return ptr->data;
     }

     Node<TYPE>* operator&()
     {
          return ptr;
```

```
	}
	bool operator == (const Node<TYPE>* tmp)
	{
		if (this->ptr == tmp)
		{
			return true;
		}
		return false;
	}
	bool operator != (const Node<TYPE>* tmp)
	{
		if (this->ptr != tmp)
		{
			return true;
		}
		return false;
	}
	Iterator& operator=(const Node <TYPE>& tmp)
	{
		if (this->ptr != tmp)
		{
			ptr = tmp;
		}
		return *this;
	}
};

****************************************************************
**********
/*StackOtmena.h
#pragma once
#include <iostream>
#include <process.h>
#include <string>
#include <conio.h>
#include <iomanip>
#include <algorithm>
#include <Windows.h>
#include <vector>
using namespace std;

template<class TYPE>
struct Value
{
	TYPE data;//данный
	Value* next;//указатель на след элемент
};
template<class TYPE>
class Stack
{
private:
	Value<TYPE>* first;
	int razmer;
```

```
public:
     Stack()
     {
          first = nullptr;
          razmer = 0;
     }
     Stack(const Stack<TYPE>& obj)
     {
          first = obj.first;
          razmer = obj.razmer;
     }
     TYPE del();
     void add_data(TYPE data);
     void pop_data();
     void ochistka();
     void show_razmer();
     ~Stack()
     {
          ochistka();
     }
};

template<class TYPE>
void Stack<TYPE>::show_razmer()
{
     cout << "Размер: " << razmer << endl;
}

template<class TYPE>
void Stack<TYPE>::add_data(TYPE data)
{
     ++razmer;
     Value<TYPE>* new_element = new Value<TYPE>;
     new_element->data = data;
     new_element->next = nullptr;
     if (!first)
     {
          first= new_element;
     }
     else
     {
          new_element->next = first;
          first = new_element;
     }
}

template<class TYPE>
void Stack<TYPE>::pop_data()
{
     if (!first)
     {
          cout << "\nСтек пустой" << endl;
          return;
     }
     Value<TYPE>* n = first;
     while (n)
     {
          cout << endl << n->data;
          n = n->next;
     }
     cout << endl;
}

template<class TYPE>
TYPE Stack<TYPE>::del()
{
     TYPE str;
     Value<TYPE>* n = first;
     first = first->next;
     str = n->data;
     delete n;
     return str;
}


template <class TYPE>
void Stack<TYPE>::ochistka()
{
     while (first != nullptr)
     {
          Value<TYPE>* n = first->next;
          delete first;
          first = n;
     }
     razmer = 0;
}

****************************************************************
*******************

/*String.h

#pragma once
#include <iostream>
#include <iomanip>
#include <conio.h>
#include <process.h>
#include <windows.h>
#include <string>
#include <sstream>
using namespace std;
class String
{
     char* mas;
     int razmer;
public:
     String() : razmer(0)
```

```
	{
		mas = nullptr;
	}
	String(const String& ss)
	{
		this->razmer = strlen(ss.mas);
		this->mas = new char[razmer + 1];
		strcpy_s(this->mas, strlen(ss.mas) + 1, ss.mas);
	}
	String(int length)
	{
		razmer = length;
		mas = new char[length + 1];
	}
	friend ostream& operator<<(ostream& on, String const& ss);
	friend istream& operator>>(istream& in, String& ss);
	String& operator=(const String& obj);
	String& operator+=(const String& obj);
	String operator+(const String& obj);
	bool operator>(const String& obj);
	bool operator<(const String& obj);
	bool operator==(String& obj);
	friend bool operator!=(const String& obj, const String& 
obj1);
	char& operator[](int s);
	char* operator = (const char* str);
	void operator = (char str[]);		// для константных 
строк
	bool operator == (const String& tmp) const;
	bool operator != (const char* tmp) const;
	bool operator ==(string str);
	/*bool operator == (const char* tmp) const;*/
	void operator += (char* str)
	{
		int len = this->razmer + strlen(str);
		char* tmp_str = new char[this->razmer + 1];
		strcpy_s(tmp_str,strlen(this->mas)+1, this->mas);
		if (this->razmer != 0)
			delete this->mas;
		mas = new char[len + 1];
		strcpy_s(mas,strlen(tmp_str)+1, tmp_str);
		strcat_s(mas,strlen(mas)+strlen(str)+1, str);
		this->razmer = len;
	}

	~String();
};


/*String.cpp

#include "String.h"

istream& operator>>(istream& in, String& ss)
```

```
{
     if(ss.mas!=NULL)
          delete[] ss.mas;
     delete[] ss.mas;
     char buf[80];
     rewind(stdin);
     in.getline(buf, sizeof buf);
     ss.razmer = strlen(buf);
     ss.mas = new char[ss.razmer + 1];
     strcpy_s(ss.mas, strlen(buf) + 1, buf);
     return in;
}
ostream& operator<<(ostream& on, const String& ss)
{
     on << ss.mas;
     return on;
}
String& String :: operator =(const String& obj)
{
     if (this != &obj)
     {
          delete[] mas;
          razmer = obj.razmer;
          mas = new char[razmer + 1];
          strcpy_s(mas, strlen(obj.mas) + 1, obj.mas);
     }
     return *this;
}

String& String::operator+=(const String& obj)
{
     *this = *this + obj;
     return *this;
}

//перегруженный оператор присваивания String - char *
char* String::operator = (const char* str)
{
     delete[] this->mas;
     this->razmer = strlen(str);
     this->mas = new char[this->razmer + 1];
     strcpy_s(this->mas,strlen(str)+1, str);
     return mas;
}
//перегруженный оператор присваивания String - char str[]
void String::operator = (char str[])
{
     delete[] this->mas;
     this->razmer = strlen(str);
     this->mas = new char[this->razmer + 1];
     strcpy_s(this->mas,strlen(str)+1, str);
}
//перегруженный оператор сравнени на равенство String - String
```

```
bool String::operator == (const String& tmp) const
{
     if ((strcmp(this->mas, tmp.mas))==0)
          return true;
     else
          return false;
}
bool String::operator == (String& tmp)
{
     if ((strcmp(this->mas, tmp.mas)) == 0)
          return true;
     else
          return false;
}
bool String::operator ==(string str)
{
     if ((strcmp(this->mas, str.c_str())) == 0)
          return true;
     else
          return false;
}
bool String::operator != (const char* tmp) const
{
     if ((strcmp(mas, tmp)) != 0)
          return true;
     else
          return false;
}
bool operator!=(const String& obj, const String& obj1)
{
     if ((strcmp(obj.mas, obj1.mas)) != 0)
          return true;
     else
          return false;
}
char& String::operator[](int s)
{
     if (s<0 || s>razmer)
     {
          cout << "Выход за пределы массива" << endl;
          exit(1);
     }
     return mas[s];
}
String String::operator+(const String& obj)
{
     String temp;

     int dlina = strlen(mas);
     int dlina1 = strlen(obj.mas);

     temp.razmer = dlina + dlina1;
     temp.mas = new char[dlina + dlina1 + 1];

     int i;
     strcpy_s(temp.mas, strlen(mas) + 1, mas);
     strcat_s(temp.mas, strlen(mas) + strlen(obj.mas) + 1,
obj.mas);
     return temp;

}
bool String::operator>(const String& obj)
{
     if ((strcmp(this->mas, obj.mas)) > 0)
          return true;
}

bool String::operator<(const String& obj)
{
     if ((strcmp(this->mas, obj.mas)) < 0)
          return false;
}

String::~String()
{
     delete[] this->mas;
}
****************************************************************

/*TXTFILE.h

#pragma once
#include <iostream>
#include <fstream>
#include "Shablon.h"
using namespace std;

class TXTFILE
{
public:
     TXTFILE() {};
     TXTFILE(string _title);
     ~TXTFILE() {};
     template<class TYPE>
     void toFile(TYPE& obj, string _filename);
     template<class TYPE>
     void fromFile(TYPE& obj, string _filename, int i);
     int checkCount(string _filename);
     void clear(string _filename);
};

template<class TYPE>
void TXTFILE::toFile(TYPE& obj, string _filename)
{
     ofstream ofs(_filename, ofstream::app);
     if (!ofs)
```

```
}

    {
        cout << "Не удалось открыть файл: " << _filename;
        system("pause");
        return;
    }
    ofs <<= obj;
    ofs << endl;
    ofs.close();
}

template<class TYPE>
void TXTFILE::fromFile(TYPE& obj, string _filename, int i)
{
    ifstream ifs(_filename, ifstream::in);
    if (!ifs)
    {
        cout << "Не удалось открыть файл: " << _filename;
        system("pause");
        return;
    }
    string s;
    while (i > 0)
    {
        getline(ifs, s);
        i--;
    }
    ifs >>= obj;
    ifs.close();
}

int TXTFILE::checkCount(string _filename)
{
    ifstream ifs(_filename, ifstream::in);
    if (!ifs)
    {
        cout << "Не удалось открыть файл: " << _filename;
        system("pause");
        return -1;
    }
    int count = 0;
    string ss;
    while (getline(ifs, ss))
    {
        count++;
    }
    ifs.close();
    return count;
}

void TXTFILE::clear(string _filename)
{
    ofstream ofs(_filename, ios::out,ios::trunc);
    ofs.close();
```

```
}
```