Министерство образования Республики Беларусь

Учреждение образования

«Гомельский государственный университет   
имени Франциска Скорины»

Факультет физики и

информационных технологий

Кафедра автоматизированных систем обработки информации

Обработка наборов данных

Курсовая работа

по учебной дисциплине

"Основы алгоритмизации и программирования"

Исполнитель

студентка группы АС-16 Лакизо М.М.

Руководитель

ст. преподаватель Дробышевский В. А.

Гомель 2021

**Содержание**

[Введение 3](#_Toc8588956)

[1 Описание комплекса программ 4](#_Toc8588957)

[1.1 Структура программного комплекса osn\_progr 4](#_Toc8588958)

[1.2 Описание модуля типов types 5](#_Toc8588959)

[1.3 Описание модуля загрузки loading\_md 6](#_Toc8588960)

[1.4 Описание модуля обработки menuproc 7](#_Toc8588961)

[1.5 Описание модуля вывода таблицы table 12](#_Toc8588962)

[2 Проверка работы на контрольном примере 15](#_Toc8588963)

[2.1 Запуск главной программы osn\_progr 15](#_Toc8588964)

[2.2 Описание работы процедуры добавление новых элементов в конец МД 15](#_Toc8588965)

[2.3 Описание работы процедуры вывода МД на экран 16](#_Toc8588966)

[2.4 Описание работы процедуры перехода к другому МД 16](#_Toc8588967)

[2.5 Описание работы процедуры поиска элемента по индексу 17](#_Toc8588968)

[2.6 Описание работы процедуры сохранения заданного элемента во втором МД 17](#_Toc8588969)

[2.7 Описание работы процедуры добавления всех элементов текущего МД в конец второго МД. 18](#_Toc8588970)

[Заключение 19](#_Toc8588971)

[Список использованных источников 20](#_Toc8588972)

[Приложение А 21](#_Toc8588973)

[Приложение Б 23](#_Toc8588974)

# Введение?

Данная курсовая работа создана для формирования списков в динамической памяти и работой с ними на языке программирования Pascal. Эта программа работает с наборами данных в динамической памяти. Множество данных представляет собой массив записей, хранящихся в динамической памяти, обращение к которым осуществляется посредством указателей.

Перечень выполняемых запросов:

1. Добавление новых элементов в конец текущего МД.
2. Вывод на дисплей всех элементов МД поэкранно.
3. Поиск элемента по номеру.
4. Переход к работе со вторым МД.
5. Создание нового МД из всех элементов с заданным значением ключевого поля.
6. Добавление всех элементов текущего МД в конец второго МД.

Множество данных – группы в детском саду.

1. шифр группы (string);
2. количество детей в группе (integer);
3. фамилия воспитателя (string);
4. тип группы: ясельная, младшая, средняя, старшая (перечислимыйтип).

Курсовая работа содержит две главы. В первой главе описан программный комплекс. Она включает в себя структуру программного комплекса и описание модулей. Вторая глава – это описание работы программы по контрольному примеру. Каждый параграф содержит реализацию одного из запросов с описанием процесса выполнения работы, а также скриншоты экрана исходного состояния записей, процесса выполнения и конечного состояния.

# Описание комплекса программ

## Структура программного комплекса osn\_progr

Данная программа является основной. Она вызывает модуль вертикального меню, модуль загрузки МД, модуль запросов, модуль типов. В соответствии с выбранным пунктом меню, с помощью стрелок на клавиатуре, программа выполняет тот или иной запрос в отношении МД.

program kyrsach;

uses crt,vertmenu,loading\_md,types,table,menuproc;

var k,i,w:integer;

mass\_str:menu\_mass = ('Adding element','Print MD',

'Swapping MD','Searching element',

'Saving element','Adding MD','Break program');

first\_link,second\_link,work,work\_link:a;

begin

k:=1;

i:=1;

w:=0;

{Creation of two data sets (MD)}

first\_link:=nil;

load\_md(first\_link,second\_link);

work\_link:=first\_link;

repeat

clrscr;

{Print curent item}

{Print head}

print\_head;

{Find curent item}

work:=work\_link;

if i<>1 then

begin

repeat

w:= w + 1;

if w=i then

break;

work:=work^.next;

until false;

print\_value\_work(i,work);

w:=0;

end

else

BEGIN

print\_value\_work(i,work\_link);

end;

{Print bot}

print\_bot;

k:= menutoscr(mass\_str,n,x,y);

case k of

{Adding new data to the end of the current MD}

1:adding\_el(work\_link,i);

{Print of all elements of MD}

2:print\_md(work\_link,i);

{Going to work with another MD}

3: begin

i:=1;

swap\_md(work\_link,first\_link,second\_link);

end;

{Search item by number}

4:search\_el\_num(work\_link,i);

{Saving the specified item in the second MD}

5:begin

if work\_link = first\_link then

saving\_el(work\_link,second\_link,i)

else

saving\_el(work\_link,first\_link,i);

{endif}

end;

{Adding all elements of the current MD to the

end of the second MD.}

6:begin

if work\_link = first\_link then

adding\_md(work\_link,second\_link)

else

adding\_md(work\_link,first\_link);

{endif}

end;

{break programm}

7:begin

clrscr;

break;

end;

end;

until false;

end.

## Описание модуля типов types

### Структура модуля типов types

Модуль содержит все типы данных, которые используются в программе. К ним относятся list – запись с данными, list2 – ссылочная запись, переменная spisok - перечислимый тип, переменная f1 описывающая тип файла. menu\_mas массив для хранения строк меню.

unit types;

{modul with types}

interface

uses crt;

const n = 7;

x = 20;

y = 10;

type

spisok = (licey,gimnazia,sr\_school);

a = ^list2;

list = record

sec\_name: string;

place: integer;

points: integer;

zaveden: spisok;

end;

list2 = record

value:list;

next:a;

end;

f1 = file of list;

menu\_mass = array[1..n] of string;

implementation

end.

## Описание модуля загрузки loading\_md

### Структура модуля загрузки loading\_md

Модуль для загрузки в память данных из файлов md1.txt и md2.txt. С помощью двух процедур new\_MD и load\_MD. Типы для переменных данного модуля хранятся в модуле types.

unit loading\_md;

{load md modul}

interface

uses table,types,crt;

{Writing to the dynamic memory of the list2 from the name\_md file}

procedure new\_MD(var start:a; name\_md:string);

{Record in dynamic memory of lists of MD1 and MD2}

procedure load\_md(var start:a; var start2:a);

implementation

{Procedures description}

end.

### Процедура new\_MD

Загружает в память данные МД из txt файла с именем name\_file и передаёт в вызывающую процедуру начало списка start.

{Writing to the dynamic memory of the list2 from the name\_md file}

procedure new\_MD(var start:a; name\_md:string);

var f:f1;

work:a;

str:list;

begin

{open file m1.txt for read}

assign(f,name\_md);

reset(f);

read(f,str);

{read in memory first first element of list}

new(start);

start^.value.sec\_name := str.sec\_name;

start^.value.place := str.place;

start^.value.points := str.points;

start^.value.zaveden := str.zaveden;

start^.next := nil;

work := start;

{read in memory next elements of list}

repeat

read(f,str);

new(work^.next);

work:=work^.next;

work^.value.sec\_name := str.sec\_name;

work^.value.place := str.place;

work^.value.points := str.points;

work^.value.zaveden := str.zaveden;

work^.next := nil;

until eof(f);

close(f);

end;

### Процедура load\_MD

Вызывает две процедуры new\_MD для загрузки данных и двух файлов m1.txt и m2.txt в память. Передаёт в вызывающую программу начало двух списков в динамической памяти start и start2.

{Record in dynamic memory of lists of MD1 and MD2}

procedure load\_md(var start:a; var start2:a);

begin

{load m1.txt}

new\_md(start,'m1.txt');

{load m2.txt}

new\_md(start2,'m2.txt');

end;

## Описание модуля обработки menuproc

### Структура модуля обработки menuproc

В модуле menuproc описаны все процедуры обработки МД.

unit menuproc;

interface

uses types,crt,table;

{Adding new data to the end of the current MD}

procedure adding\_el(start:a; var index\_current\_value:integer);

{Print of all elements of MD}

procedure print\_md(work:a;var index\_current\_value:integer);

{Going to work with another MD}

procedure swap\_md(var work\_link:a; first\_md,second\_md:a);

{Search item by number}

procedure search\_el\_num(start:a; var index\_current\_value:integer);

{Saving the specified item in the second MD}

procedure Saving\_el(start1:a;start2:a; index\_current\_value:integer);

{Adding all elements of the current MD to the end of the second MD}

procedure Adding\_md(start1:a;start2:a);

implementation

{Procedures description}

End.

### Процедура adding\_el

Процедура adding\_el добавляет новый эл-т заданный с клавиатуры, в конец текущего МД. Возвращает индекс добавленного эл-та index\_current\_value.

{Adding new data to the end of the current MD}

procedure adding\_el(start:a; var index\_current\_value:integer);

var n,w,i:integer;

work:a;

begin

{Input and initialization}

clrscr;

i:=0;

w:=1;

readln(n);

work:=start;

{Going to the end of md}

repeat

w:= w + 1;

work:= work^.next;

until work^.next = nil;

clrscr;

repeat

{Adding element}

i:= i + 1;

new(work^.next);

work:=work^.next;

{Input sec\_name, place, points for new element}

{error point in case of error}

repeat

{$i-}

{Input zaveden for new element}

if IOResult<>0 then

{Incorrect input message}

else break;

{$i+}

until false;

work^.next:=nil;

clrscr;

until i = n;

{Print Table}

print\_body(start,n);

{Retern current value index}

index\_current\_value:=w + i;

end;

### Процедура print\_md

Процедура print\_md выводит все элементы текущего МД на экран в таблице и возвращает индекс последнего эл-та в главную программу index\_current\_value.

{Print of all elements of MD}

procedure print\_md(work:a;var index\_current\_value:integer);

var i:integer;

begin

clrscr;

i:=0;

print\_body(work,i);

index\_current\_value:=i;

end;

### Процедура search\_el\_num

Процедура search\_el\_num находит заданный элемент в текущем МД начиная с указателя start и возвращает индекс найденного эл-та в главную программу index\_current\_value.

{Search item by number}

procedure search\_el\_num(start:a; var index\_current\_value:integer);

var find:integer;

i:integer;

work:a;

begin

clrscr;

i:=0;

readln(find);

work:=start;

{Finding the right item}

repeat

i:= i + 1;

if find = i then

break;

{endif}

work:=work^.next;

until work = nil;

if work = nil then

{Message about empty element}

else

begin

{Print found item}

print\_head();

print\_value\_work(find,work);

print\_bot();

{Return current value index}

index\_current\_value:=find;

end;

{endif}

end;

### Процедура swap\_md

Процедура swap\_MD меняет текущий МД на другой МД. В указатель work\_link записывается указатель на другой МД и передаётся в вызывающую программу.

{Going to work with another MD}

procedure swap\_md(var work\_link:a; first\_md,second\_md:a);

begin

clrscr;

{Swap m1 and m2}

if work\_link = first\_md then

work\_link:=second\_md

else

work\_link:=first\_md;

{endif}

end;

### Процедура saving\_el

Процедура saving\_el сохраняет заданный элемент текущего МД в конец второго. Указателем на начало текущего МД служит start1, на конец второго – start2.

{Saving the specified item in the second MD}

procedure Saving\_el(start1:a;start2:a; index\_current\_value:integer);

var i,find:integer;

work,work2:a;

begin

clrscr;

i:=0;

readln(find);

work:=start1;

{Finding the right item}

repeat

i:= i + 1;

if find = i then

break;

{endif}

work:=work^.next;

until work = nil;

{Message about empty link}

if work = nil then

{Message about empty element}

else

begin

work2:=start2;

{Going to the end of the second md}

repeat

work2:=work2^.next

until work2^.next=nil;

{Saving the specified item in the second MD}

new(work2^.next);

work2:=work2^.next;

work2^.value:=work^.value;

work2^.next:=nil;

end;

{endif}

end;

### Процедура adding\_md

Процедура adding\_md добавляет текущий МД в конец второго. Указателем на начало текущего МД служит start1 другого start2.

{Adding all elements of the current MD to the end of the second MD}

procedure Adding\_md(start1:a;start2:a);

var work1,work2,swap1,swap2:a;

begin

clrscr;

work1:=start1;

work2:=start2;

new(swap1);

swap1^.value:=start1^.value;

swap2:=swap1;

{Copy current md}

repeat

if work1^.next = nil then break;

new(swap2^.next);

swap2:=swap2^.next;

work1:=work1^.next;

swap2^.value:=work1^.value;

swap2^.next:=nil

until false;

{Going to the end of the second md}

repeat

work2:=work2^.next

until work2^.next=nil;

{Adding all elements of the current MD to the end of the second MD}

work2^.next:=swap1;

end;

## Описание модуля вывода таблицы table

### Структура модуля вывода таблицы table

В модуле table описаны все процедуры вывода таблицы и элементов текущего МД на экран.

unit table;

interface

uses crt,types;

{Print md in the table}

procedure print\_body(work:a; var i:integer);

{Print table's head}

procedure print\_head;

{Print work value}

procedure print\_value\_work(i:integer;work:a);

{Print table's bot}

procedure print\_bot;

implementation

{Procedures description}

end.

### Процедура print\_head

Процедура print\_head выводит шапку таблицы на экран.

{Print table's head}

procedure print\_head;

begin

{Table's head}

end;

### Процедура print\_bot

Процедура print\_bot выводит низ таблицы на экран.

{Print table's bot}

procedure print\_bot;

begin

{Table's bot}

end;

### Процедура print\_value\_work

Процедура print\_value\_work выводит поля с информацией на экран.

{Print work value}

procedure print\_value\_work(i:integer;work:a);

var x:integer;

begin

{Print the number of string}

{Print sec\_name}

{Alignment sec\_name}

if 20>length(work^.value.sec\_name) then

for x:=1 to 20-length(work^.value.sec\_name) do

{Alignment}

{endfor x}

{endif}

{Print Place}

{Print Points}

{Print zaveden}

case work^.value.zaveden of

licey:

{Lyceum}

gimnazia:

{Gimnazium}

sr\_school:

{Sec.school}

end;

end;

### Процедура print\_body

Процедура print\_body поэкранно выводит данные МД по 20 элементов. Начиная с указателя work. После завершения процедуры она возвращает ин-декс текущего элемента i в вызывающую программу.

{Print md in the table}

procedure print\_body(work:a; var i:integer);

begin

clrscr;

{Print Head}

print\_head;

i:=0;

{Print body}

repeat

i:= i + 1;

if i=20 then

begin

print\_value\_work(i,work);

print\_bot;

work:=work^.next;

print\_head;

continue;

end;

{Print punkt with number i}

print\_value\_work(i,work);

{move to the next item}

work:=work^.next;

until work=nil;

{Print bot}

print\_bot;

end;

# Проверка работы на контрольном примере

## Запуск главной программы osn\_progr

После запуска программа загружает в память данные из двух файлов m1.txt и m2.txt с помощью процедуры load\_MD и ждёт ввода любого символа (рисунок 1).

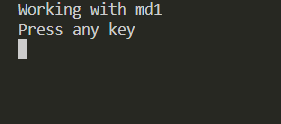


Рисунок 1 – Окно программы после запуска

## Описание работы процедуры добавление новых элементов в конец МД

Для того чтобы создать добавить новый элемент в конец текущего набора данных нужно выбрать в меню «Adding element». После нажатия клавиши Enter, программа запросит ввод данных для нового элемента. Затем требуется нажать любую клавишу (рисунок 2). Затем выводится измененные данные текущего МД в виде таблицы (рисунок 3).

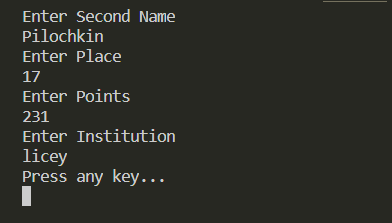


Рисунок 2 – Ввод информации

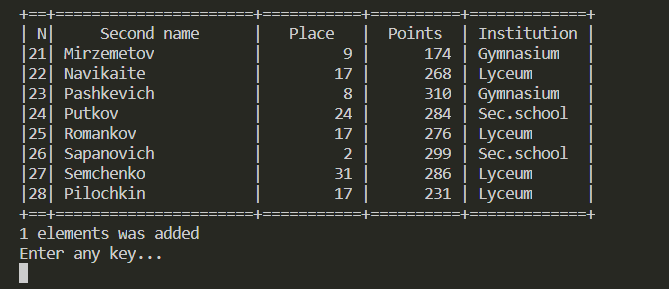


Рисунок 3 – Таблица с данными

## Описание работы процедуры вывода МД на экран

Для вывода таблицы с данными списка нужно выбрать пункт меню «Print md».

После этого на экран выводится таблица с данными (рисунок 4).

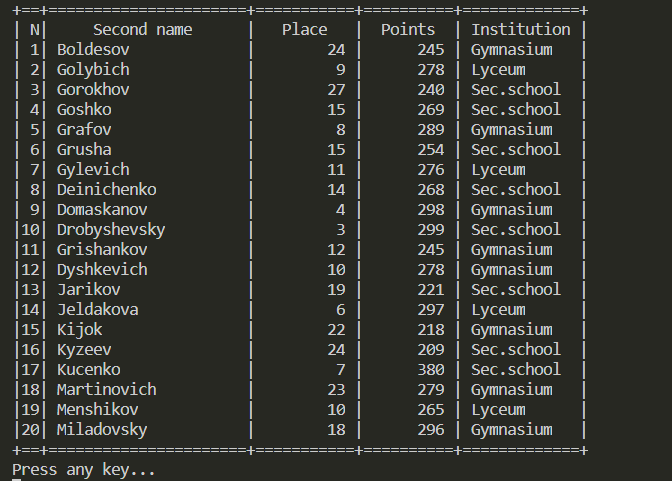


Рисунок 4 – Вывод таблицы с данными

## Описание работы процедуры перехода к другому МД

Для перехода к другому МД следует выбрать пункт меню «Swapping MD» и нажать Enter, после чего текущий МД смениться на второй. Проверить это можно с помощью 2-го пункта меню (рисунок 5).

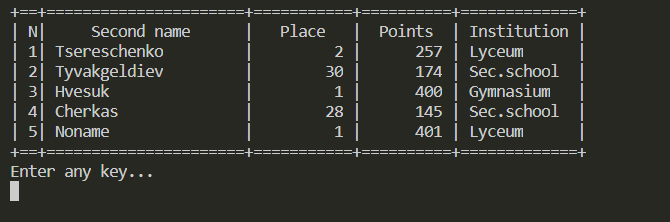


Рисунок 5 – Переход к работе с другим МД

## Описание работы процедуры поиска элемента по индексу

Для поиска элемента по его индексу нужно выбрать пункт меню «Searching element». Далее требуется ввести нужный индекс. И затем выводится найденный элемент (рисунок 6).

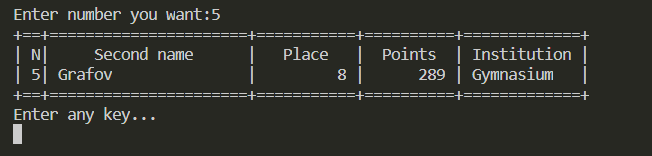


Рисунок 6 – Поисковый запрос

## Описание работы процедуры сохранения заданного элемента во втором МД

Для сохранения заданного элемента во втором МД нужно выбрать пункт меню «Savind element»(рисунок 7). После выбора (рисунок 8) произойдёт копирование выбранного вами элемента во второй МД (рисунок 9).

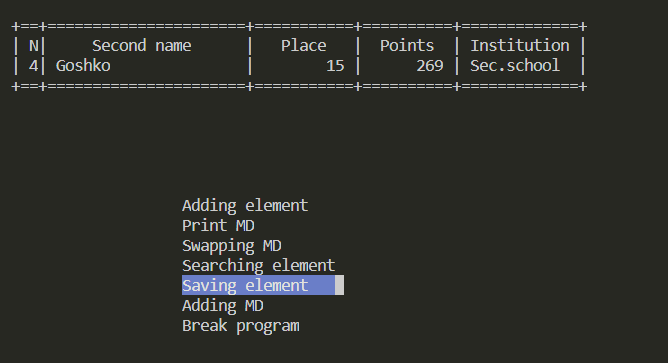


Рисунок 7 – Выбор пункта и обзор текущего элемента

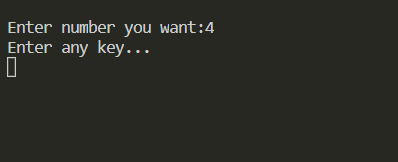


Рисунок 8 – Выполнение запроса

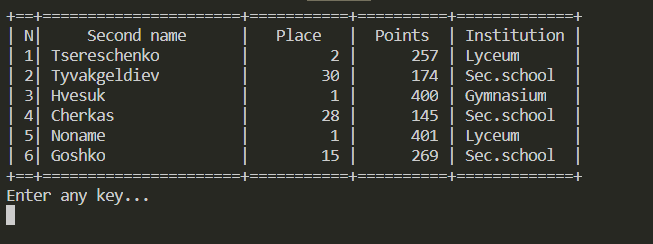


Рисунок 9 – Результат

## Описание работы процедуры добавления всех элементов текущего МД в конец второго МД.

При выборе пункта «Adding MD in current MD» данные текущего МД добавятся к концу второго (рисунок 10).

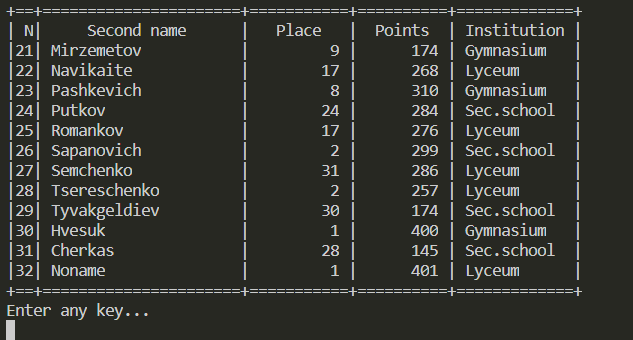


Рисунок 10 – Результат

# Заключение

В ходе выполнения курсовой работы была написана и протестирована программа, работающая с МД. Программа обрабатывает МД в динамической памяти. Само МД представляет собой записи, обращение к которым осуществляется посредством указателей.

Программа реализовывает следующие запросы:

1. Добавление новых элементов в конец текущего МД.
2. Вывод на дисплей всех элементов МД поэкранно.
3. Поиск элемента по номеру.
4. Переход к работе со вторым МД.
5. Создание нового МД из всех элементов с заданным значением ключевого поля.
6. Добавление всех элементов текущего МД в конец второго МД.

Множество данных – группы в детском саду.

1. шифр группы (string);
2. количество детей в группе (integer);
3. фамилия воспитателя (string);
4. тип группы: ясельная, младшая, средняя, старшая (перечислимыйтип).

Программа удовлетворяет следующим условиям и ограничениям:

1. Главная процедура программы реализует работу с вертикальным меню.
2. Процедуры, реализующие запросы, должны быть размещены в одном или более модулях.
3. Никаких глобальных переменных использовать нельзя.
4. Вместе с меню на экране должно быть отображено:
   1. Номер текущего элемента,
   2. Содержимое текущего элемента.
5. На экран выводить элементы в виде таблицы (один элемент – одна строка таблицы).
6. Если после выполнения запроса изменяется хотя бы один элемент, то заканчивать запрос выводом всего множества элементов.
7. Тестами к заданиям служат два НД с правдоподобной информацией, подготовленные на внешнем носителе.

# Список использованных источников

1. Е.А.Зуев «Программирование на языке Turbo Pascal»
2. Т.А.Павловская «Паскаль»
3. А.В.Осипов «Pascal ABC.NET»
4. E.P.Алексеев «Турбо Паскаль 7.0»

# Приложение А

Текст основной программы

program kyrsach;

uses crt,vertmenu,loading\_md,types,table,menuproc;

var k,i,w:integer;

mass\_str:menu\_mass = ('Adding element','Print MD',

'Swapping MD','Searching element',

'Saving element','Adding MD','Break program');

first\_link,second\_link,work,work\_link:a;

begin

k:=1;

i:=1;

w:=0;

{Creation of two data sets (MD)}

first\_link:=nil;

load\_md(first\_link,second\_link);

work\_link:=first\_link;

repeat

clrscr;

{Print curent item}

{Print head}

print\_head;

{Find curent item}

work:=work\_link;

if i<>1 then

begin

repeat

w:= w + 1;

if w=i then

break;

work:=work^.next;

until false;

print\_value\_work(i,work);

w:=0;

end

else

BEGIN

print\_value\_work(i,work\_link);

end;

{Print bot}

print\_bot;

k:= menutoscr(mass\_str,n,x,y);

case k of

{Adding new data to the end of the current MD}

1:adding\_el(work\_link,i);

{Print of all elements of MD}

2:print\_md(work\_link,i);

{Going to work with another MD}

3: begin

i:=1;

swap\_md(work\_link,first\_link,second\_link);

end;

{Search item by number}

4:search\_el\_num(work\_link,i);

{Saving the specified item in the second MD}

5:begin

if work\_link = first\_link then

saving\_el(work\_link,second\_link,i)

else

saving\_el(work\_link,first\_link,i);

{endif}

end;

{Adding all elements of the current MD to the

end of the second MD.}

6:begin

if work\_link = first\_link then

adding\_md(work\_link,second\_link)

else

adding\_md(work\_link,first\_link);

{endif}

end;

{break programm}

7:begin

clrscr;

break;

end;

end;

writeln('Enter any key...');

readln();

until false;

end.

# Приложение Б

Текст модуля loading\_md

unit loading\_md;

{load md modul}

interface

uses table,types,crt;

{Writing to the dynamic memory of the list2 from the name\_md file}

procedure new\_MD(var start:a; name\_md:string);

{Record in dynamic memory of lists of MD1 and MD2}

procedure load\_md(var start:a; var start2:a);

implementation

{Writing to the dynamic memory of the list2 from the name\_md file}

procedure new\_MD(var start:a; name\_md:string);

var f:f1;

work:a;

str:list;

begin

{open file m1.txt for read}

assign(f,name\_md);

reset(f);

read(f,str);

{read in memory first first element of list}

new(start);

start^.value.sec\_name := str.sec\_name;

start^.value.place := str.place;

start^.value.points := str.points;

start^.value.zaveden := str.zaveden;

start^.next := nil;

work := start;

{read in memory next elements of list}

repeat

read(f,str);

new(work^.next);

work:=work^.next;

work^.value.sec\_name := str.sec\_name;

work^.value.place := str.place;

work^.value.points := str.points;

work^.value.zaveden := str.zaveden;

work^.next := nil;

until eof(f);

close(f);

end;

{Record in dynamic memory of lists of MD1 and MD2}

procedure load\_md(var start:a; var start2:a);

begin

{load m1.txt}

new\_md(start,'m1.txt');

{load m2.txt}

new\_md(start2,'m2.txt');

writeln('Working with md1');

writeln('Press any key');

readln();

end;

end.

Текст модуля menuproc

unit menuproc;

interface

uses types,crt,table;

{Adding new data to the end of the current MD}

procedure adding\_el(start:a; var index\_current\_value:integer);

{Print of all elements of MD}

procedure print\_md(work:a;var index\_current\_value:integer);

{Going to work with another MD}

procedure swap\_md(var work\_link:a; first\_md,second\_md:a);

{Search item by number}

procedure search\_el\_num(start:a; var index\_current\_value:integer);

{Saving the specified item in the second MD}

procedure Saving\_el(start1:a;start2:a; index\_current\_value:integer);

{Adding all elements of the current MD to the end of the second MD.}

procedure Adding\_md(start1:a;start2:a);

implementation

{Adding new data to the end of the current MD}

procedure adding\_el(start:a; var index\_current\_value:integer);

var n,w,i:integer;

work:a;

begin

{Input and initialization}

clrscr;

i:=0;

w:=1;

write('How many elements do you want to add?');

readln(n);

work:=start;

{Going to the end of md}

repeat

w:= w + 1;

work:= work^.next;

until work^.next = nil;

clrscr;

repeat

{Adding element}

i:= i + 1;

new(work^.next);

work:=work^.next;

{Input sec\_name, place, points for new element}

writeln('Enter Second Name');

readln(work^.value.sec\_name);

writeln('Enter Place');

readln(work^.value.place);

writeln('Enter Points');

readln(work^.value.points);

{error point in case of error}

repeat

{$i-}

{Input zaveden for new element}

writeln('Enter Institution');

readln(work^.value.zaveden);

if IOResult<>0 then

begin

{Incorrect input message}

writeln('Error input, enter correct value:');

writeln('licey,gimnazia,sr\_school')

end

else break;

{$i+}

until false;

work^.next:=nil;

Writeln('Press any key...');

readln();

clrscr;

until i = n;

{Print Table}

print\_body(start,n);

writeln(i,' elements was added');

{Retern current value index}

index\_current\_value:=w + i;

end;

{Print of all elements of MD}

procedure print\_md(work:a;var index\_current\_value:integer);

var i:integer;

begin

clrscr;

i:=0;

print\_body(work,i);

index\_current\_value:=i;

end;

{Going to work with another MD}

procedure swap\_md(var work\_link:a; first\_md,second\_md:a);

begin

clrscr;

writeln('Swapping md...');

{Swap m1 and m2}

if work\_link = first\_md then

work\_link:=second\_md

else

work\_link:=first\_md;

{endif}

end;

{Search item by number}

procedure search\_el\_num(start:a; var index\_current\_value:integer);

var find:integer;

i:integer;

work:a;

begin

clrscr;

i:=0;

Write('Enter number you want:');

readln(find);

work:=start;

{Finding the right item}

repeat

i:= i + 1;

if find = i then

break;

{endif}

work:=work^.next;

until work = nil;

if work = nil then

{Message about empty element}

writeln('Nothing found')

else

begin

{Print found item}

print\_head();

print\_value\_work(find,work);

print\_bot();

{Return current value index}

index\_current\_value:=find;

end;

{endif}

end;

{Saving the specified item in the second MD}

procedure Saving\_el(start1:a;start2:a; index\_current\_value:integer);

var i,find:integer;

work,work2:a;

begin

clrscr;

i:=0;

Write('Enter number you want:');

readln(find);

work:=start1;

{Finding the right item}

repeat

i:= i + 1;

if find = i then

break;

{endif}

work:=work^.next;

until work = nil;

{Message about empty link}

if work = nil then

{Message about empty element}

writeln('Nothing found')

else

begin

work2:=start2;

{Going to the end of the second md}

repeat

work2:=work2^.next

until work2^.next=nil;

{Saving the specified item in the second MD}

new(work2^.next);

work2:=work2^.next;

work2^.value:=work^.value;

work2^.next:=nil;

end;

{endif}

end;

{Adding all elements of the current MD to the end of the second MD.}

procedure Adding\_md(start1:a;start2:a);

var work1,work2,swap1,swap2:a;

begin

clrscr;

writeln('Adding MD');

work1:=start1;

work2:=start2;

new(swap1);

swap1^.value:=start1^.value;

swap2:=swap1;

{Copy current md}

repeat

if work1^.next = nil then break;

new(swap2^.next);

swap2:=swap2^.next;

work1:=work1^.next;

swap2^.value:=work1^.value;

swap2^.next:=nil

until false;

{Going to the end of the second md}

repeat

work2:=work2^.next

until work2^.next=nil;

{Adding all elements of the current MD to the end of the second MD}

work2^.next:=swap1;

end;

end.

Текст модуля unit types

unit types;

{modul with types}

interface

uses crt;

const n = 7;

x = 20;

y = 10;

type

spisok = (licey,gimnazia,sr\_school);

a = ^list2;

list = record

sec\_name: string;

place: integer;

points: integer;

zaveden: spisok;

end;

list2 = record

value:list;

next:a;

end;

f1 = file of list;

menu\_mass = array[1..n] of string;

implementation

end.

Текст модуля table

unit table;

interface

uses crt,types;

{Print md in the table}

procedure print\_body(work:a; var i:integer);

{Print table's head}

procedure print\_head;

{Print work value}

procedure print\_value\_work(i:integer;work:a);

{Print table's bot}

procedure print\_bot;

implementation

{Print table's head}

procedure print\_head;

begin

{Table's head}

write('+==+======================+===========+’);

writeln(‘==========+=============+');

write('| N| Second name | Place |’);

writeln(‘Points | Institution |');

end;

{Print table's bot}

procedure print\_bot;

begin

{Table's bot}

write('+==+======================+===========’);

writeln(‘+==========+=============+');

end;

{Print work value}

procedure print\_value\_work(i:integer;work:a);

var x:integer;

begin

{Print the number of string}

if i <= 9 then

write('| ',i,'| ')

else

write('|',i,'| ');

{endif}

{Print sec\_name}

write(work^.value.sec\_name);

{Alignment sec\_name}

if 20>length(work^.value.sec\_name) then

for x:=1 to 20-length(work^.value.sec\_name) do

{Aligment}

write(' ');

{endfor x}

{Print Place}

write(' |',work^.value.place:10,' |');

{Print Points}

write(work^.value.points:9,' |');

{Print zaveden}

case work^.value.zaveden of

licey:

{Lyceum}

write(' Lyceum ');

gimnazia:

{Gimnazium}

write(' Gymnasium ');

sr\_school:

{Sec.school}

write(' Sec.school ');

end;

writeln('|');

end;

{Print md in the table}

procedure print\_body(work:a; var i:integer);

begin

clrscr;

{Print Head}

print\_head;

i:=0;

{Print body}

repeat

i:= i + 1;

if i=20 then

begin

print\_value\_work(i,work);

print\_bot;

work:=work^.next;

writeln('Press any key...');

readln();

print\_head;

continue;

end;

{Print punkt with number i}

print\_value\_work(i,work);

{move to the next item}

work:=work^.next;

until work=nil;

{Print bot}

print\_bot;

end;

end.