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](#_Toc71772517)

[1 Описание комплекса программ 4](#_Toc71772518)

[1.1 Структура программного комплекса main 4](#_Toc71772519)

[1.2 Описание модуля типов types 5](#_Toc71772520)

[1.3 Описание модуля загрузки loading\_md 6](#_Toc71772521)

[1.4 Описание модуля обработки procedures 8](#_Toc71772522)

[1.5 Описание модуля вывода таблицы table 12](#_Toc71772523)

[2 Проверка работы на контрольном примере 15](#_Toc71772524)

[2.1 Запуск главной программы main 15](#_Toc71772525)

[2.2 Описание работы процедуры добавление новых элементов в конец МД 15](#_Toc71772526)

[2.3 Описание работы процедуры вывода МД на экран 16](#_Toc71772527)

[2.4 Описание работы процедуры перехода к другому МД 16](#_Toc71772528)

[2.5 Описание работы процедуры поиска элемента по индексу 17](#_Toc71772529)

[2.6 Описание работы процедуры сохранения заданного элемента во втором МД 17](#_Toc71772530)

[2.7 Описание работы процедуры добавления всех элементов текущего МД в конец второго МД. 18](#_Toc71772531)

[Заключение 20](#_Toc71772532)

[Список использованных источников 21](#_Toc71772533)

[Приложение А 22](#_Toc71772534)

[Приложение Б 24](#_Toc71772535)

# 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. Сохранение заданного элемента во втором МД.
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 main

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 main;

uses crt, print\_table, types, menu, procedures, load\_md;

var k, i, w:integer;

str\_menu\_array:menu\_array = ('Add new element', 'Print current MD',

'Fing element', 'Swap MD', 'Save specified element to the second md',

'Add all elements of curent MD to the end of 2nd MD', 'Exit program');

start\_1,start\_2,work,work\_link:a;

begin

k:=1;

i:=1;

w:=0;

{Creation of two data sets (MD)}

start\_1:=nil;

load\_mds(start\_1,start\_2);

work\_link:=start\_1;

repeat

clrscr;

{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\_row(i,work);

w:=0;

end

else

begin

print\_row(i,work\_link);

end;

{Print bot}

print\_bot;

{Menu implementation}

k:=print\_menu(str\_menu\_array, menu\_fields\_num, x, y);

case k of

{Add new element to the end of the curent MD}

1:add\_new\_el(work\_link, i);

{Print all elements of the curent MD }

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

{Find element by index}

3:find\_el(work\_link, i);

{Swap working MD}

4: begin

i := 1;

swap\_md(work\_link, start\_1, start\_2);

end;

{Saving the specified item in the second MD}

5: begin

if work\_link = start\_1 then

saving\_el(work\_link,start\_2,i)

else

saving\_el(work\_link,start\_1,i);

{endif}

end;

{Add all elements from the current MD to the end of the 2nd MD}

6: begin

if work\_link = start\_1 then

adding\_all\_to\_second\_md(work\_link, start\_2)

else

adding\_all\_to\_second\_md(work\_link, start\_1);

{endif}

end;

{Exit program}

7: begin

clrscr;

break;

end;

end;

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

readln();

until false;

end.

## Описание модуля типов types

### Структура модуля типов types

Модуль содержит все типы данных, которые используются в программе. К ним относятся data – запись с данными, data\_node – ссылочная запись, переменная group\_type\_enum - перечислимый тип, переменная file\_type описывающая тип файла. Menu\_array массив для хранения строк меню.

unit types;

{module with types}

interface

uses crt;

const menu\_fields\_num = 7;

x = 20;

y = 10;

type

menu\_array = array[1..menu\_fields\_num] of string;

group\_type\_enum = (yaselnaya, mladshaya, srednaya, starshaya);

a = ^data\_node;

data = record

group\_code: string;

children\_number: integer;

teacher\_name: string;

group\_type: group\_type\_enum;

end;

data\_node = record

value: data;

next: a;

end;

file\_type = file of data;

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 load\_md;

{create md on physical memory and load it to dynamic memory}

interface

uses types, crt;

{create md}

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

{load md}

procedure load\_mds(var start\_1:a; var start\_2: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 list from the name\_md file}

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

var f:file\_type;

work:a;

buffer:data;

begin

{open file with name name\_md for read}

assign(f,name\_md);

reset(f);

read(f,buffer);

{read in memory first first element of list}

new(start);

start^.value.group\_code := buffer.group\_code;

start^.value.children\_number := buffer.children\_number;

start^.value.teacher\_name := buffer.teacher\_name;

start^.value.group\_type := buffer.group\_type;

start^.next := nil;

work := start;

{read in memory next elements of list}

repeat

read(f,buffer);

new(work^.next);

work:=work^.next;

work^.value.group\_code := buffer.group\_code;

work^.value.children\_number := buffer.children\_number;

work^.value.teacher\_name := buffer.teacher\_name;

work^.value.group\_type := buffer.group\_type;

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 start1 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\_1,'m1.txt');

{load m2.txt}

new\_md(start\_2,'m2.txt');

end;

## Описание модуля обработки procedures

### Структура модуля обработки menuproc

В модуле procedures описаны все процедуры обработки МД.

unit procedures;

interface

uses types, crt, print\_table;

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

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

{Print of all elements of MD}

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

{Find item by number}

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

{Going to work with another MD}

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

{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\_all\_to\_second\_md(start1:a;start2:a);

implementation

{Procedures description}

End.

### Процедура add\_new\_el

Процедура add\_new\_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 add\_new\_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 group\_code, children\_number, teacher\_name for new element}

writeln('Enter Group code');

readln(work^.value.group\_code);

writeln('Enter Children number');

readln(work^.value.children\_number);

writeln('Enter Teacher name');

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

{error point in case of error}

repeat

{$i-}

{Input group type for new element}

writeln('Enter Group type');

readln(work^.value.group\_type);

if IOResult<>0 then

begin

{Incorrect input message}

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

writeln('yaselnaya, mladshaya, srednaya, starshaya')

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;

### Процедура print\_current\_md

Процедура print\_current \_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\_current \_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 find\_el

Процедура find\_el 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 find\_el(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;

writeln('Swapping md...');

{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\_all\_to\_second\_md

Процедура adding\_all\_to\_second\_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\_all\_to\_second\_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;

## Описание модуля вывода таблицы table

### Структура модуля вывода таблицы print\_table

В модуле print\_table описаны все процедуры вывода таблицы и элементов текущего МД на экран.

unit print\_table;

interface

uses crt, types;

{Print tables head}

procedure print\_head;

{Print table's bot}

procedure print\_bot;

{Print one row}

procedure print\_row(i:integer;work:a);

{Print body}

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

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\_row

Процедура print\_row выводит поля с информацией на экран.

{Print one row}

procedure print\_row(i:integer;work:a);

var x:integer;

begin

{Print the number of string}

{Print group\_code }

{Alignment group\_code }

if 12>length(work^.value.group\_code) then

for x:=1 to 12-length(work^.value. group\_code) do

{Alignment}

{endfor x}

{endif}

{Print children\_number}

{Print teacher\_name}

{Print group\_type}

case work^.value.group\_type of

yaselnaya:

{ yaselnaya };

mladshaya:

{ mladshaya };

srednaya:

{ srednaya };

starshaya:

{ starshaya };

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 body}

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

begin

clrscr;

{Print Head}

print\_head;

i:=0;

{Print body}

repeat

i:= i + 1;

if I mod 20 = 0 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 main

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

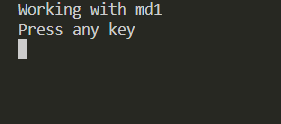


Рисунок 1 – Окно программы после запуска

## 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 «Add new 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).

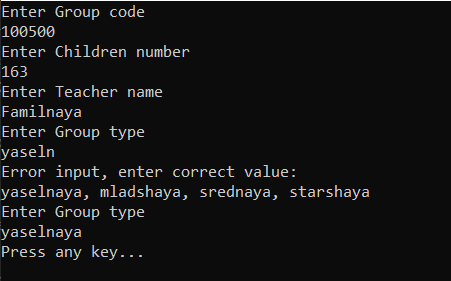


Рисунок 2 – Ввод информации

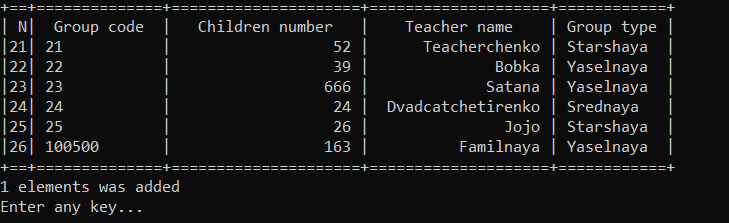


Рисунок 3 – Таблица с данными

## Описание работы процедуры вывода МД на экран

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

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

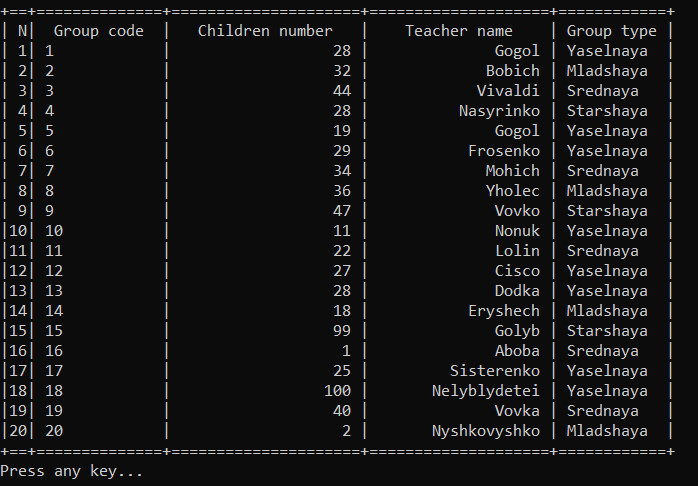


Рисунок 4 – Вывод таблицы с данными

## Описание работы процедуры перехода к другому МД

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

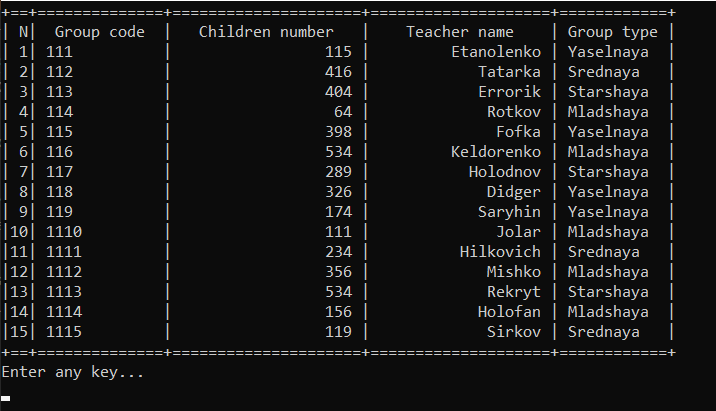


Рисунок 5 – Переход к работе с другим МД

## Описание работы процедуры поиска элемента по индексу

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

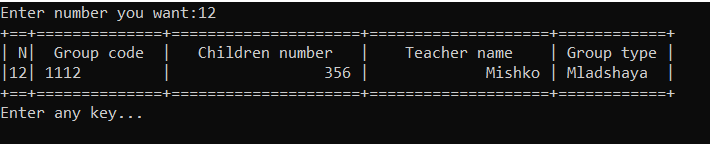


Рисунок 6 – Поисковый запрос

## Описание работы процедуры сохранения заданного элемента во втором МД

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

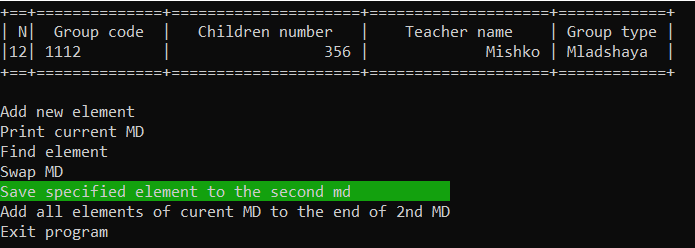


Рисунок 7 – Выбор пункта и обзор текущего элемента

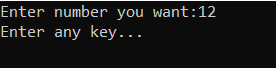


Рисунок 8 – Выполнение запроса

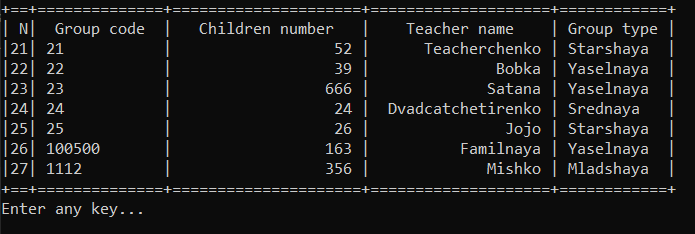


Рисунок 9 – Результат

## 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).

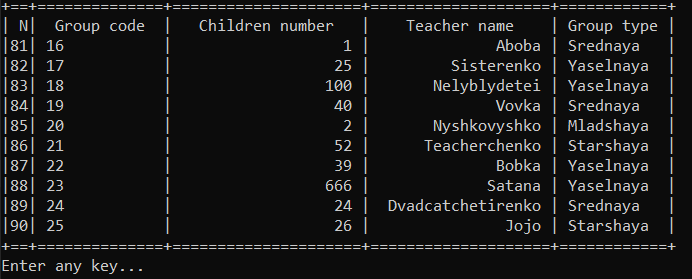


Рисунок 10 – Результат

# 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. Сохранение заданного элемента во втором МД.
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 (перечислимый тип).

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. Зуев, Е. А. Программирование на языке Turbo Pascal 6.0,7.0./ Е.А. Зу-ев. – М.: М-Веста, 1993.
2. Павловская, Т. А. Паскаль. Программирование на языке высокого уровня./ Т. А. Павловская. – СПб.: Питер, 2004.
3. А.В.Осипов, Pascal ABC.NET 3.5/ А.В.Осипов. – М.: Ростов-на-Дону, 2019.
4. E.P.Алексеев, Турбо Паскаль 7.0/ E.P.Алексеев. — М.: НТ Пресс, 2009.

# Приложение А

Текст основной программы

program main;

uses crt, print\_table, types, menu, procedures, load\_md;

var k, i, w:integer;

str\_menu\_array:menu\_array = ('Add new element', 'Print current MD',

'Fing element', 'Swap MD', 'Save specified element to the second md',

'Add all elements of curent MD to the end of 2nd MD', 'Exit program');

start\_1,start\_2,work,work\_link:a;

begin

k:=1;

i:=1;

w:=0;

{Creation of two data sets (MD)}

start\_1:=nil;

load\_mds(start\_1,start\_2);

work\_link:=start\_1;

repeat

clrscr;

{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\_row(i,work);

w:=0;

end

else

begin

print\_row(i,work\_link);

end;

{Print bot}

print\_bot;

{Menu implementation}

k:=print\_menu(str\_menu\_array, menu\_fields\_num, x, y);

case k of

{Add new element to the end of the curent MD}

1:add\_new\_el(work\_link, i);

{Print all elements of the curent MD }

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

{Find element by index}

3:find\_el(work\_link, i);

{Swap working MD}

4: begin

i := 1;

swap\_md(work\_link, start\_1, start\_2);

end;

{Saving the specified item in the second MD}

5: begin

if work\_link = start\_1 then

saving\_el(work\_link,start\_2,i)

else

saving\_el(work\_link,start\_1,i);

{endif}

end;

{Add all elements from the current MD to the end of the 2nd MD}

6: begin

if work\_link = start\_1 then

adding\_all\_to\_second\_md(work\_link, start\_2)

else

adding\_all\_to\_second\_md(work\_link, start\_1);

{endif}

end;

{Exit program}

7: begin

clrscr;

break;

end;

end;

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

readln();

until false;

end.

# Приложение Б

Текст модуля load\_md

unit load\_md;

{create md on physical memory and load it to dynamic memory}

interface

uses types, crt;

{create md}

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

{load md}

procedure load\_mds(var start\_1:a; var start\_2:a);

implementation

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

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

var f:file\_type;

work:a;

buffer:data;

begin

{open file with name name\_md for read}

assign(f,name\_md);

reset(f);

read(f,buffer);

{read in memory first first element of list}

new(start);

start^.value.group\_code := buffer.group\_code;

start^.value.children\_number := buffer.children\_number;

start^.value.teacher\_name := buffer.teacher\_name;

start^.value.group\_type := buffer.group\_type;

start^.next := nil;

work := start;

{read in memory next elements of list}

repeat

read(f,buffer);

new(work^.next);

work:=work^.next;

work^.value.group\_code := buffer.group\_code;

work^.value.children\_number := buffer.children\_number;

work^.value.teacher\_name := buffer.teacher\_name;

work^.value.group\_type := buffer.group\_type;

work^.next := nil;

until eof(f);

close(f);

end;

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

procedure load\_mds(var start\_1:a; var start\_2:a);

begin

{load m1.txt}

new\_md(start\_1,'m1.txt');

{load m2.txt}

new\_md(start\_2,'m2.txt');

writeln('Workink with md1');

writeln('Press any key');

readln();

end;

end.

Текст модуля procedures

unit procedures;

interface

uses types, crt, print\_table;

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

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

{Print of all elements of MD}

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

{Find item by number}

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

{Going to work with another MD}

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

{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\_all\_to\_second\_md(start1:a;start2:a);

implementation

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

procedure add\_new\_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 group\_code, children\_number, teacher\_name for new element}

writeln('Enter Group code');

readln(work^.value.group\_code);

writeln('Enter Children number');

readln(work^.value.children\_number);

writeln('Enter Teacher name');

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

{error point in case of error}

repeat

{$i-}

{Input group type for new element}

writeln('Enter Group type');

readln(work^.value.group\_type);

if IOResult<>0 then

begin

{Incorrect input message}

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

writeln('yaselnaya, mladshaya, srednaya, starshaya')

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

var i:integer;

begin

clrscr;

i:=0;

print\_body(work,i);

index\_current\_value:=i;

end;

{Find item by number}

procedure find\_el(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\_row(find,work);

print\_bot();

{Return current value index}

index\_current\_value:=find;

end;

{endif}

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;

{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\_all\_to\_second\_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 types

unit types;

{module with types}

interface

uses crt;

const menu\_fields\_num = 7;

x = 1;

y = 6;

type

menu\_array = array[1..menu\_fields\_num] of string;

group\_type\_enum = (yaselnaya, mladshaya, srednaya, starshaya);

a = ^data\_node;

data = record

group\_code: string;

children\_number: integer;

teacher\_name: string;

group\_type: group\_type\_enum;

end;

data\_node = record

value: data;

next: a;

end;

file\_type = file of data;

implementation

end.

Текст модуля print\_table

unit print\_table;

interface

uses crt, types;

{Print tables head}

procedure print\_head;

{Print table's bot}

procedure print\_bot;

{Print one row}

procedure print\_row(i:integer;work:a);

{Print body}

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

implementation

{Print tables head}

procedure print\_head;

begin

{Tables head}

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

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

writeln('| N| Group code | Children number | Teacher name | Group type |');

end;

{Print table's bot}

procedure print\_bot;

begin

{Table's bot}

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

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

end;

{Print one row}

procedure print\_row(i:integer;work:a);

var x:integer;

begin

{Print the number of string}

if i <= 9 then

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

else

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

{endif}

{Print group\_code}

write(work^.value.group\_code);

{Alignment group\_code}

if 12>length(work^.value.group\_code) then

for x:=1 to 12-length(work^.value.group\_code) do

write(' ');

{endfor x}

{endif}

{Print children\_number}

write(' |',work^.value.children\_number:20,' |');

{Print teacher\_name}

write(work^.value.teacher\_name:19,' |');

{Print group\_type}

case work^.value.group\_type of

yaselnaya:

write(' Yaselnaya ');

mladshaya:

write(' Mladshaya ');

srednaya:

write(' Srednaya ');

starshaya:

write(' Starshaya ');

end;

writeln('|');

end;

{Print body}

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

begin

clrscr;

{Print Head}

print\_head;

i:=0;

{Print body}

repeat

i:= i + 1;

if i mod 20 = 0 then

begin

print\_row(i,work);

print\_bot;

work:=work^.next;

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

readln();

print\_head;

continue;

end;

{Print punkt with number i}

print\_row(i,work);

{move to the next item}

work:=work^.next;

until work=nil;

{Print bot}

print\_bot;

end;

end.