Министерство образования и науки Российской Федерации

Севастопольский государственный университет

Кафедра ИС

Отчет

по лабораторной работе № 3

«Программирование с использованием

объектов и модулей»

Выполнил

ст. гр. И12д

Серегин А.В.

Проверил:

асс. Забаштанский А.К.

Севастополь

2015

1. Цель работы

Изучение базовых понятий ООП, приобретение навыков разработки и отладки программ, использующих модули и объекты. Исследование особенностей поисковых бинарных деревьев, представленных в виде объекта.

2. Варианты заданий

Создать модуль, содержащий описание объекта, который представляет бинарное дерево в соответствии с вариантом из предыдущей лабораторной работы. Объект должен обладать возможностью добавления новых элементов, удаления существующих, поиска элемента по ключу, обхода дерева, а также выполнять дополнительные операции в соответствии с вариантом задания.

Написать программу для представления таблицы, заданной в предыдущей лабораторной работе, в виде бинарного дерева. Программа должна содержать меню, позволяющее проверить все методы объекта. Предусмотреть возможность ввода данных из файла и с клавиатуры.

Таблица 1 - Расписание

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № Поезда | Станция отправки | Станция назначения | Время отправки | Время прибытия | Стоимость билета |

Вариант 20. Таблица 3.2. Процедуру, которая подсчитывает число вершин на n-ом уровне непустого дерева T.

3. Структурная схема

Ниже представлены структурные схемы алгоритма работы программы.



Рисунок 1 – функция NewTrain.

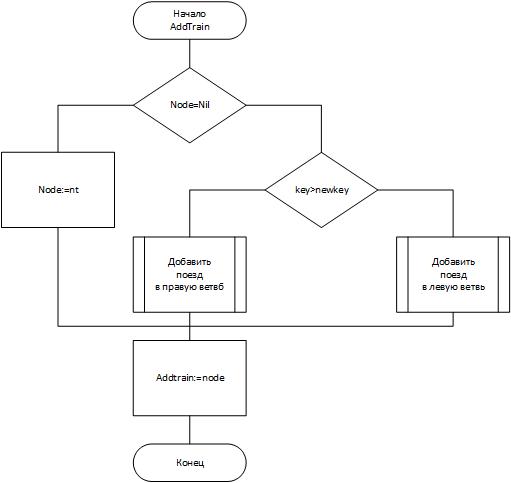


Рисунок 2 – процедура AddTrain.



Рисунок 3 – процедура PrintTrainFile.



Рисунок 4 – процедура DeleteTrainTree.

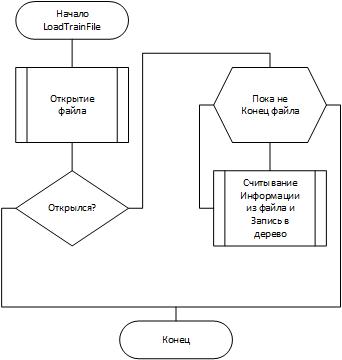


Рисунок 5 – процедура TrainTree.LoadTrainFile.

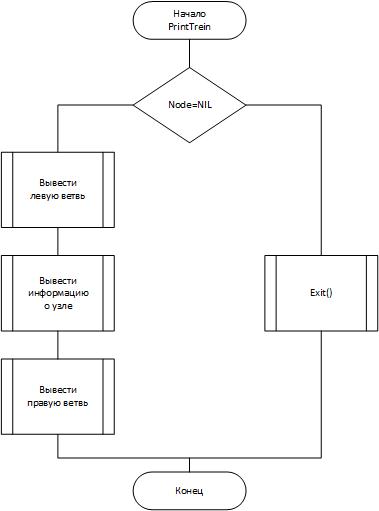


Рисунок 6 – процедура PrintTrain.

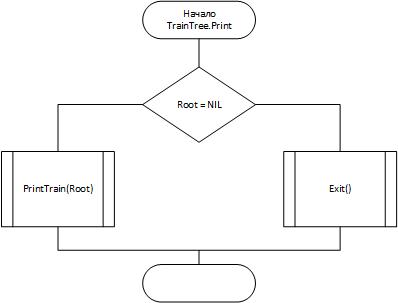


Рисунок 7 – процедура TrainTree.Print.

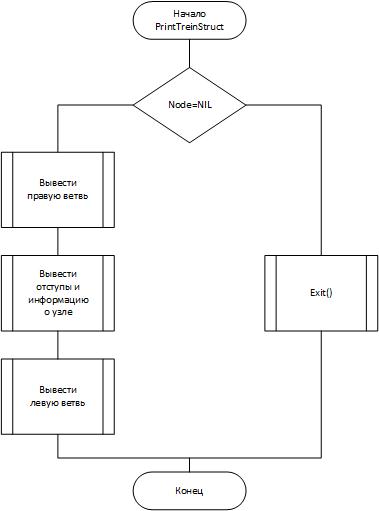


Рисунок 8 – процедура PrintTrainStruct.

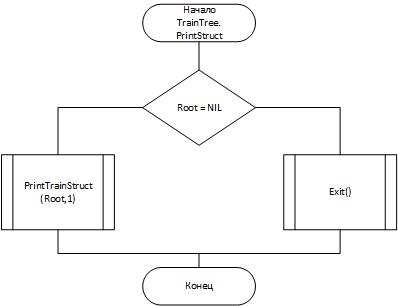


Рисунок 9 – процедура PrintStruct.

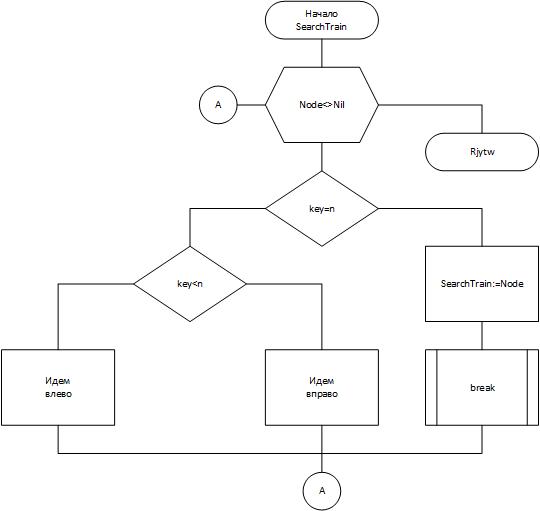


Рисунок 10 – функция SearchTrain.



Рисунок 11 – Процедура TrainTree.Add.

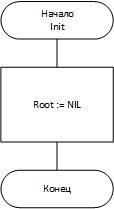
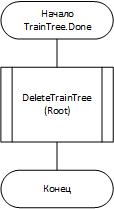
 

Рисунок 12 – конструктор и деструктор.

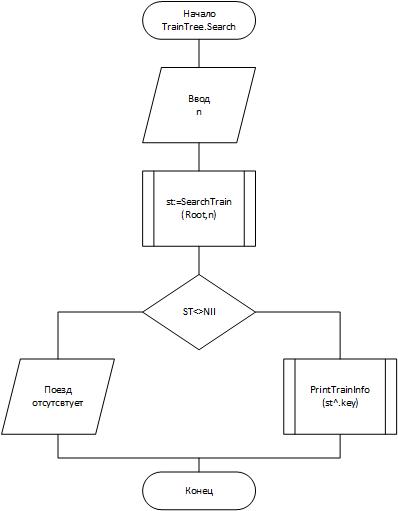


Рисунок 13 – процедура TrainTree.Search.



Рисунок 14 – процедура SaveTrainFile.



Рисунок 15 – процедура PrintTrainInfo.

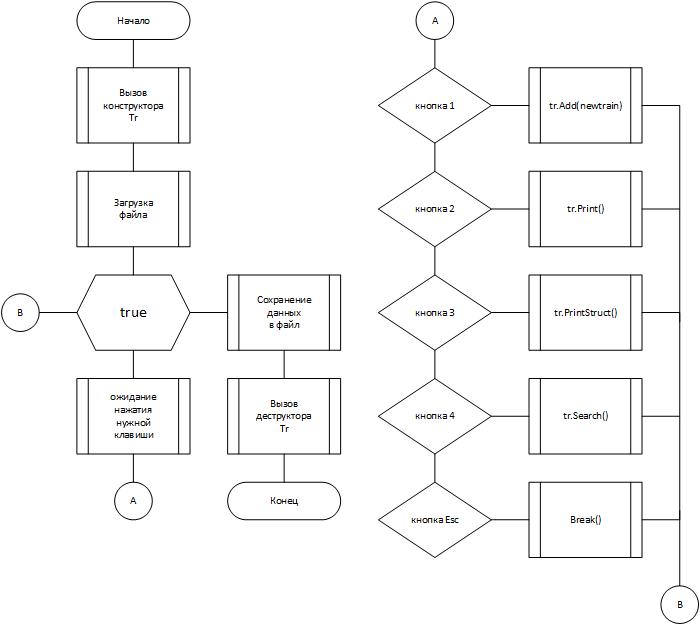


Рисунок 16 – программа.

4. Код программы

Код программы:

Program lab3\_20;

uses Train, Crt;

var tr:traintree; {дерево поездов}

f:trainfile; {файл с инф. о поездах}

key:char; {клавиша}

Begin

tr.Init(); {вызов конструктора}

assign(f,'Train.dat'); {подключение файла}

tr.LoadTrainFile(f);

while (true) do

begin

clrscr;

writeln('1.Add 2.Show 3.ShowStruct 4.Search 5.NodeCount Esc.Exit');

key:=' ';

{выбор нужной клавиши}

while (not (key in ['1','2','3','4','5',#27])) do key:=readkey;

case key of

'1': tr.Add(NewTrain());

'2': tr.Print();

'3': tr.PrintStruct();

'4': tr.Search();

'5': tr.Count();

#27: break;

end;

end;

tr.SaveTrainFile(f); {сохранение файла}

tr.Done(); {вызов деструктора}

End.

Код модуля:

Unit Train;

Interface

uses crt;

Type

int = longint;

Time = record {структура время}

hour:byte;

minute:byte;

second:byte;

end;

TrainInfo = record {структура информации}

number:int; {о поезде}

StartPoint:string[20];

EndPoint:string[20];

StartTime:Time;

EndTime:Time;

TicketPrice:int;

end;

TrainFile = File of TrainInfo; {файл с поезд.инф}

PTrain = ^Tree; {\* на узел дерева}

Tree = record {описание узла дерева}

key:TrainInfo;

Left:PTrain;

Right:PTrain;

end;

TrainTree = object {описание класса дерева поездов}

constructor Init(); {конструктор}

procedure Add(key:TrainInfo);{добавление узла}

procedure Print(); {вывод списка}

procedure PrintStruct();{вывод структуры дерева}

procedure LoadTrainFile(var f:trainfile);{загрузка файла}

Procedure SaveTrainFile(var f:trainfile);{сохр. файла}

Procedure Search(); {поиск поезда в дереве}

procedure Count(); {кол-во поездов на N уровне}

destructor Done(); {деструктор}

private

Root:PTrain; {корень дерева}

end;

function NewTrain():TrainInfo; {создание новой записи}

Implementation

Function NodeCount(root:ptrain; Level:Integer):integer; {подсчет k узлов n уровня}

var n:integer;

Begin

n:=0;

If (Level>=1) and (root<>Nil) Then

Begin

If Level=1 Then N:=N+1;

n:=n+NodeCount(root^.Left,Level-1); {переход на след. уровень}

n:=n+NodeCount(root^.Right,Level-1);

End;

NodeCount:=n;

End;

{поиск поезда в дереве по ключу}

function SearchTrain(Node:Ptrain; number:int):Ptrain;

begin

while (Node <> NIL) do

begin

if ((Node^.key.number = number) or (Node = NIL)) then

begin {нашли ключ}

SearchTrain:= Node;

break;

end

else {идем по дереву}

if (Node^.key.number < number) then

Node := Node^.Right

else

Node := Node^.Left;

end;

end;

function NewTrain():TrainInfo; {создание новой записи}

begin

clrscr();

writeln('Введите номер поезда.');

readln(NewTrain.number);

writeln('Введите место отправки.');

readln(NewTrain.StartPoint);

writeln('Введите место прибытия.');

readln(NewTrain.EndPoint);

writeln('Введите время отправки.');

readln(NewTrain.StartTime.hour, NewTrain.StartTime.minute);

writeln('Введите время прибытия.');

readln(NewTrain.EndTime.hour, NewTrain.EndTime.minute);

writeln('Введите стоимость билета.');

readln(NewTrain.TicketPrice);

end;

{вывод данных ключа на экран}

procedure PrintTrainInfo(key:TrainInfo);

Begin

write(key.number);

write(key.StartPoint:11,key.EndPoint:11);

write(key.StartTime.hour:4,':',key.StartTime.minute);

write(key.EndTime.hour:5,':',key.EndTime.minute);

writeln(key.TicketPrice:10);

End;

{добавление дерева в пустое место}

function AddTrain(Node,nt:PTrain):Ptrain;

Begin

if (Node = NIL) then {вставка в пустое место}

Node := nt

else {идем по дереву}

if (Node^.key.number > nt^.key.number) then

Node^.Left := AddTrain(Node^.Left,nt)

else

Node^.Right := AddTrain(Node^.Right,nt);

AddTrain := Node;

End;

{вывод дерева слева на право}

procedure PrintTrain(Node:Ptrain);

Begin

if (Node = NIL) then Exit();

PrintTrain(Node^.Left);

PrintTrainInfo(Node^.key);

PrintTrain(Node^.Right);

End;

{вывод структуры дерева на экран}

procedure PrintTreeStruct(Node:Ptrain; level:int);

var i:int;

Begin

if (Node <> NIL) then

begin

PrintTreeStruct(Node^.Right,level+1);

for i:=1 to level do write(' ');

writeln(Node^.key.number);

PrintTreeStruct(Node^.Left,level+1);

end;

End;

{очистка памяти}

Procedure DeleteTrainTree(Node:PTrain);

Begin

if (Node = NIL) then exit();

DeleteTrainTree(Node^.Left);

DeleteTrainTree(Node^.Right);

Dispose(Node);

End;

{запись данных в файл}

procedure PrintTrainFile(var f:trainfile; Node:ptrain);

Begin

if (Node=NIL) then exit();

write(f,Node^.key);

printtrainfile(f,Node^.left); {идем по дереву}

printtrainfile(f,Node^.right);

End;

{конструктор класса}

Constructor TrainTree.Init();

Begin

Root := NIL;

End;

{создание нового поезда}

procedure TrainTree.Add(key:TrainInfo);

var nt:Ptrain;

Begin

new(nt);

nt^.key := key;

nt^.left := NIL;

nt^.right := NIL;

Root:=AddTrain(Root,nt); {отправка в дерево}

End;

{вывод списка дерева}

procedure TrainTree.Print();

Begin

clrscr();

if (Root = NIL) then {проверка наличия дерева}

begin

writeln('Список поездов пуст.');

readkey();

exit();

end;

writeln('№','StPoint':11,'EndPoint':11,'StTime':7,'EndTime':8,'Price':10);

PrintTrain(Root); {вывод слева на право}

readkey();

End;

{вывод структуры дерева}

procedure TrainTree.PrintStruct();

Begin

clrscr();

if (Root = NIL) then {проверка наличия дерева}

begin

writeln('Список поездов пуст.');

readkey();

exit();

end;

writeln('Структура дерева поездов:');

PrintTreeStruct(Root,1); {вывод структуры дерева}

readkey();

End;

{деструктор класса}

Destructor TrainTree.Done();

Begin

DeleteTrainTree(Root);

End;

{сохранение данных в файл}

Procedure TrainTree.SaveTrainFile(var f:trainfile);

begin

rewrite(f);

PrintTrainFile(f,root); {запись данных}

close(f);

end;

{загрузка данных из файла}

procedure TrainTree.LoadTrainFile(var f:trainfile);

var nt:TrainInfo;

begin

{$I-}

reset(f);

{$I+}

if (IOResult=0) and (FileSize(f)<>0) then

while not eof(f) do

begin

read(f,nt); {считывание и добавление}

Add(nt); {данных в дерево}

end;

end;

{вывод кол-ва поздов n уровня}

procedure TrainTree.Count();

var n:int;

Begin

clrscr();

writeln('Введите номер уровня.');

readln(n);

writeln('На уровне ',n,' ',NodeCount(root,n),' узeл(ов)');

readkey();

End;

{Поиск поезда по его номеру}

procedure TrainTree.Search();

var ST:Ptrain; n:int;

Begin

clrscr();

writeln('Введите номер поезда');

readln(n);

clrscr();

ST:= SearchTrain(Root,n); {поиск указателя на структуру}

if (ST <> NIL) then

begin {вывод поезда на экран}

writeln('№','StPoint':11,'EndPoint':11,'StTime':7,'EndTime':8,'Price':10);

PrintTrainInfo(ST^.key);

end

else writeln('Поезд не найден.');

readkey;

End;

Begin

End.

5. Тестовые примеры.

Пример вывода меню изображен на Рисунке 17.

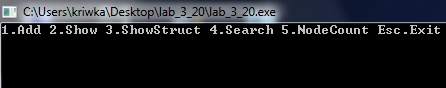


Рисунок 17 – Главное меню.

На Рисунке 18 изображен ввод информации о поезде. Сначала мы добавим поезд с номером 2, потом 1 и 3, которые пойдут в левую и правую ветвь соответственно.

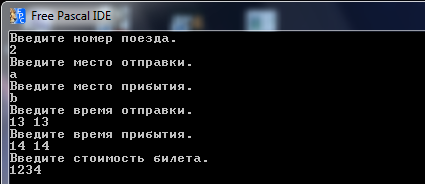


Рисунок 18 – Ввод данных.

На Рисунке 19 отображаются данные введенные нами раньше, выведенные слева направо дерева.

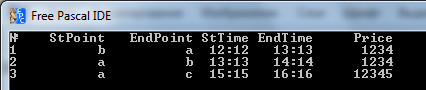


Рисунок 19 – Отображение дерева.

На Рисунке 20 показан подсчет количества узлов N уровня. Узлы 1 и 3 пошли влево и вправо от корня 2, поэтому на 2м уровне 2 узла.

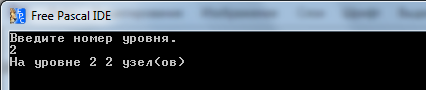


Рисунок 20 – Подсчет узлов N уровняю.

Выводы

С помощью ООП можно легко создавать свои классы(объекты), описывать различные структуры данных и с легкостью управлять ими. Сложностью ООП для человека является сложность описания классов, понимание наследование и полиморфизма. Также на языке Pascal достаточно неудобный синтаксис описания классов, отсутствие многих возможностей ООП, таких как перегрузки функций, операций, и т. д. Как по мне, ООП на паскале стоит изучать только в ознакомительных целях.