9 ЛАБОРАТОРНАЯ РАБОТА №9

«ПРОГРАММИРОВАНИЕ НЕЛИНЕЙНЫХ СТРУКТУР ДАННЫХ»

9.1 Цель работы

Исследование нелинейных структур данных и приобретение навыков разработки и отладки программ, использующих древовидные структуры. Исследование особенностей работы с поисковыми бинарными деревьями.

9.2 Вариант задания – 20

Требуется представить таблицу 9.1 в виде бинарного дерева. Написать процедуры создания и обхода дерева, а также процедуру, которая определяет, входит ли элемент в дерево. Значения полей и количество записей в таблице студент выбирает самостоятельно. Программа должна сохранять дерево в файле и создавать его заново при её повторном запуске.

Таблица 9.1 – Расписание

№ Поезда	Станция отправления	Станция назначения	Время отправления	Время прибытия	Стоимость билета

9.3 Порядок выполнения работы

9.3.1 Разработать алгоритм решения задачи, разбив его на процедуры.

9.3.2 Разработать структурную схему алгоритма решения задачи.

9.3.3 Разработать программу на языке Pascal.

9.3.4 Разработать тестовые примеры.

9.3.5 Выполнить отладку программы.

9.3.6 Сделать выводы по проделанной работе.

9.4 Ход работы

9.4.1 Для задания был разработан алгоритм решения задачи:

Был подключен модуль «crt», чтобы организовать удобную работу пользователя с программой. Так же для удобства пользователя в основном блоке

программы выводится надпись «что делает программа», а также работа с программой осуществлена с помощью оператора выбора «case», который поможет выбирать нужные действия которые должна выполнять программа. Программа может выполнить 7 задач, 6 из которых – наши процедуры и одна встроенная процедура выхода из программы. Первая задача – это процедура создания дерева, в которой автоматически удаляется предыдущая если такова имелась. Если вводится символ ‘*’ вместо станции отправления, то считывание узлов дерева заканчивается. Во второй процедуре происходит удаление дерева. Третья процедура выводит дерево на экран с поворотом на 90 градусов влево. Четвёртая узнаёт входит ли элемент в дерево. Пятая процедура сохраняет дерево в файл, для того чтобы потом можно было вернуться к сохранённому виду дерева. Шестая процедура выгружает дерево из файла для работы с ней.

9.4.2 Разработана структурная схема алгоритма решения задачи и представлена на рисунках 9.1, 9.2, 9.3.

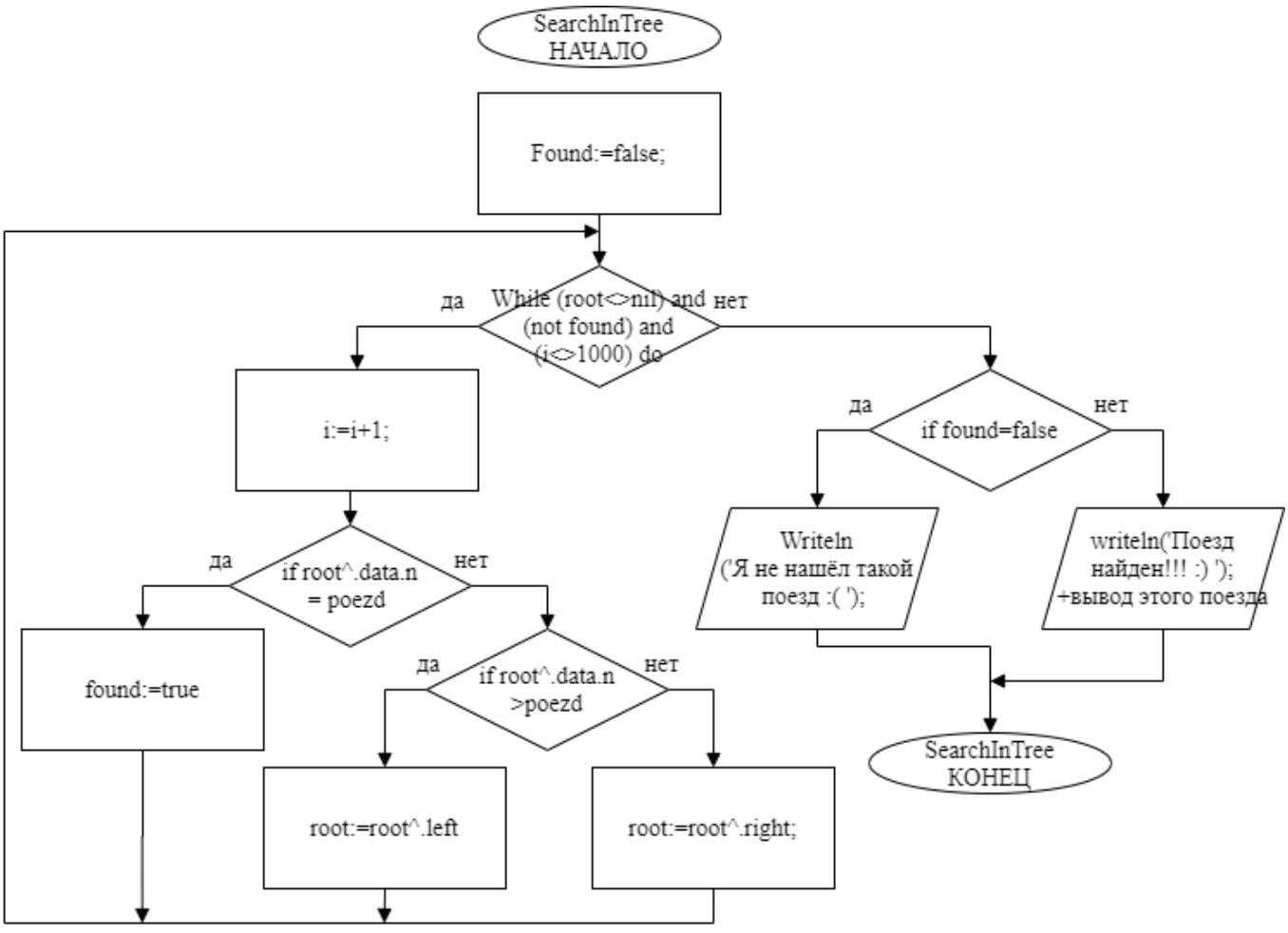

Рисунок 9.1 – Структурная схема процедуры поиска элемента в дереве

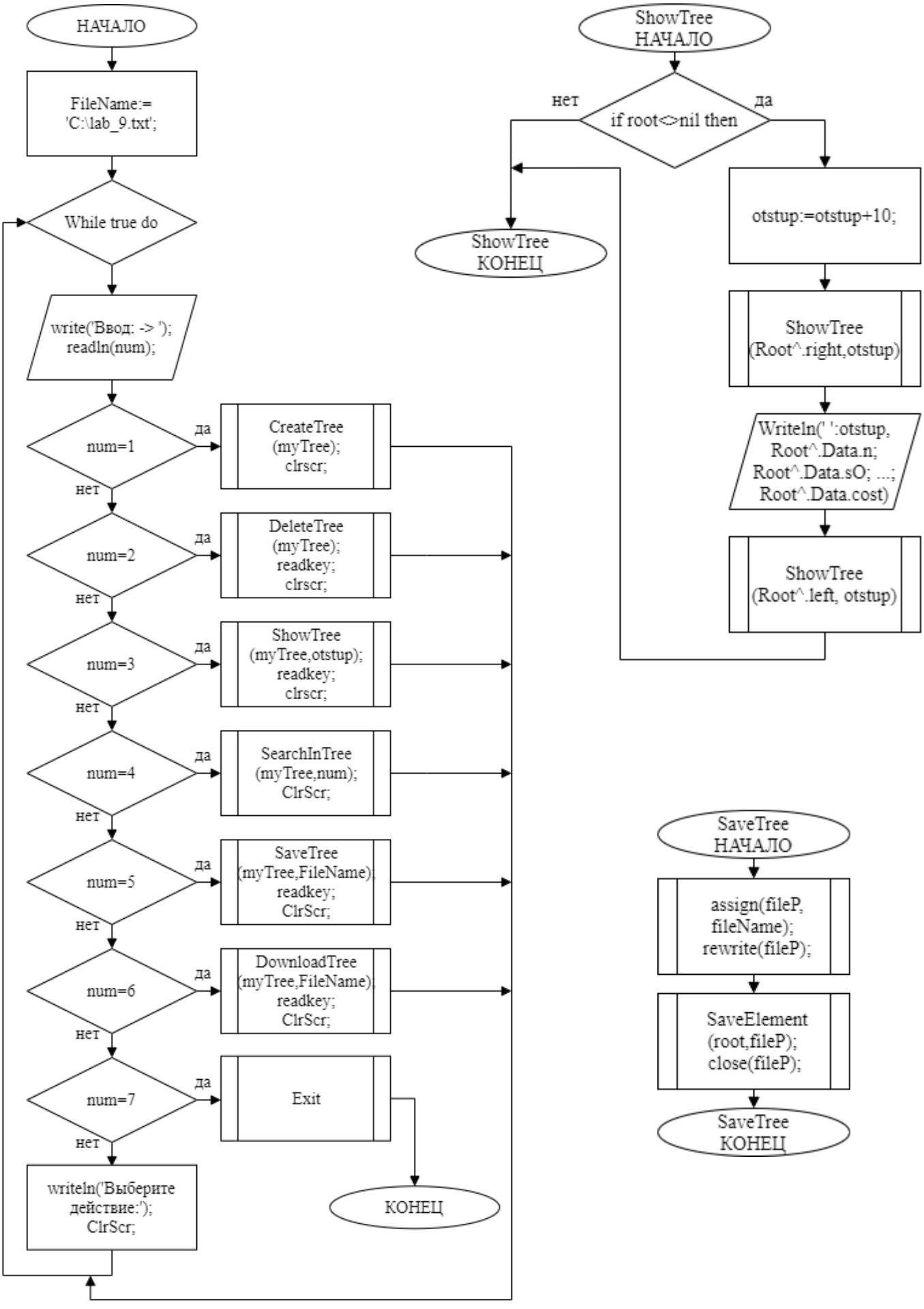

Рисунок 9.2 – Структурные схемы основной программы, процедуры демонстрации дерева и процедуры сохранения дерева

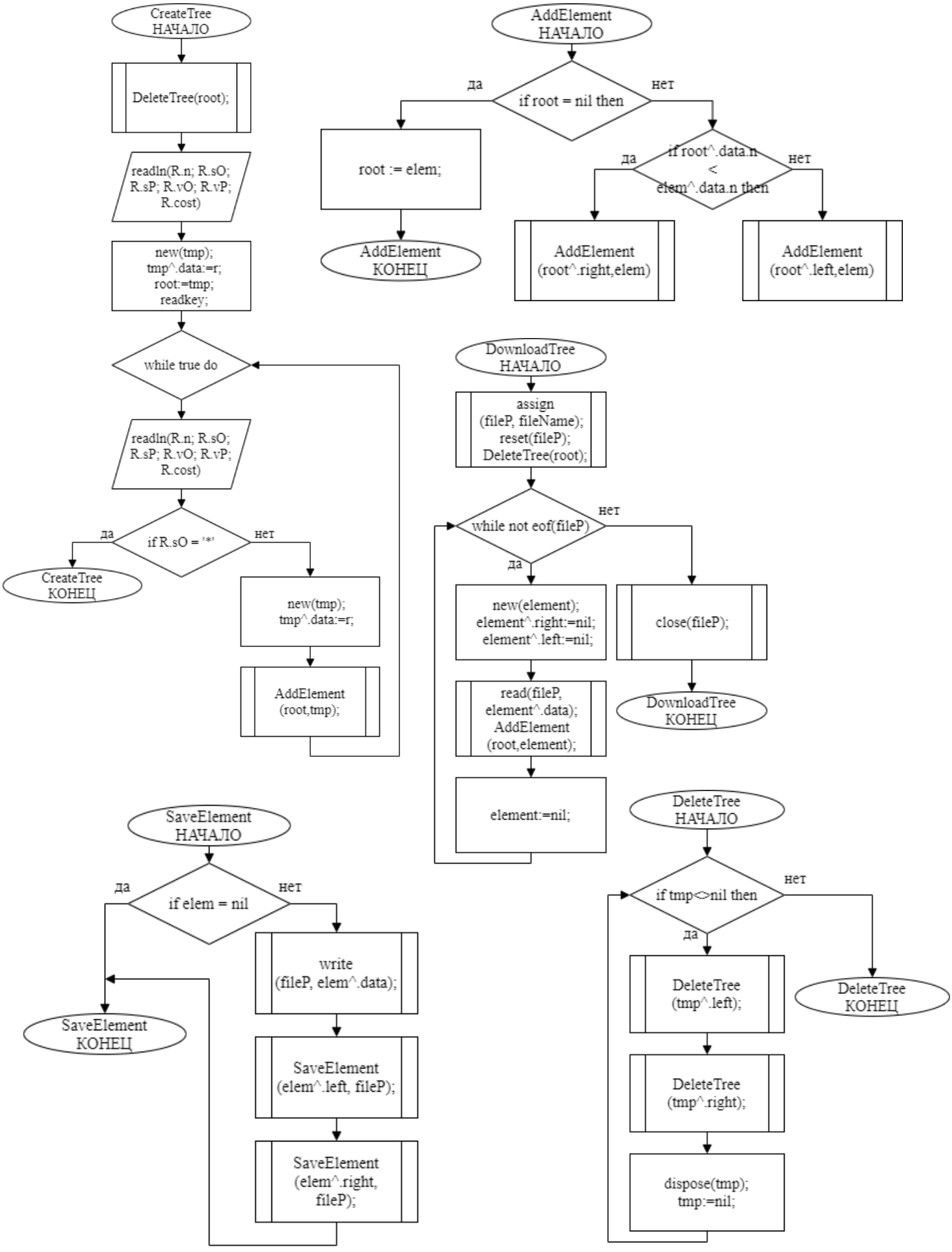

Рисунок 9.3 – Структурные схемы процедур создания дерева, добавления элемента в дерево, загрузки дерева, удаления дерева и сохранения одного элемента

9.4.3 Написана программа согласно вышеописанного алгоритма.

```
Program lab_9;
Uses crt;
Type
  rasp=record
    n,cost:integer;      //номер поезда,стоимость билета
    sO,sP:string[20];  //станция отправления/прибытия
    vO,vP:string[5];  //время отправления/прибытия
  end;

  tree=record
    data:rasp;
    left,right:^tree;
  end;

////  Удаление дерева  \\\\
Procedure DeleteTree(var tmp:^tree);
Begin
  if tmp<>nil then
  begin
    // Очистка левой и правой ветви
    DeleteTree(tmp^.left);
    DeleteTree(tmp^.right);
    // Удаление корня
    dispose(tmp);
    tmp:=nil;
  end;
End;

////  Добавить новый элемент  \\\\
Procedure AddElement(var root:^tree; const elem:^tree);
begin
  if root = nil then
  begin
    root := elem;
    EXIT;
  end;
  if root^.data.n < elem^.data.n then
    AddElement(root^.right,elem)
  else
    AddElement(root^.left,elem);
end;

////  Процедура создания дерева  \\\\
Procedure CreateTree (var root:^tree);
Var tmp:^tree; r:rasp;
Begin
  //удалить предыдущее дерево если таково было
  DeleteTree(root);

  Writeln('Создаём дерево');
  writeln;
  write('Введите номер поезда: ');         readln(R.n);
  write('Введите станцию отправления: '); readln(R.sO);
  write('Введите станцию прибытия: ');     readln(R.sP);
  write('Введите время отправления: ');    readln(R.vO);
```

```
  write('Введите время прибытия: ');      readln(R.vP);
  write('Введите стоимость билета: ');    readln(R.cost);
  new(tmp);
  tmp^.data:=r;
  root:=tmp;
  readkey;

  while true do
  begin
    clrscr;
    write('Введите номер поезда: ');        readln(R.n);
    write('Введите станцию отправления (*-выход): '); readln(R.sO);
    if R.sO = '*' then Exit;
    write('Введите станцию прибытия: ');    readln(R.sP);
    write('Введите время отправления: ');   readln(R.vO);
    write('Введите время прибытия: ');      readln(R.vP);
    write('Введите стоимость билета: ');    readln(R.cost);
    new(tmp);
    tmp^.data:=r;
    AddElement(root,tmp);
    readkey;
  end;
End;

////  Процедура демонстрации дерева  \\\\
Procedure ShowTree (const root:^tree; otstup:integer);
begin
  if root<>nil then
  Begin
    otstup:=otstup+10;
    ShowTree(Root^.right,otstup);
    Writeln(' ':otstup, Root^.Data.n);
    Writeln(' ':otstup, Root^.Data.sO);
    Writeln(' ':otstup, Root^.Data.sP);
    Writeln(' ':otstup, Root^.Data.vO);
    Writeln(' ':otstup, Root^.Data.vP);
    Writeln(' ':otstup, Root^.Data.cost);
    ShowTree(Root^.left, otstup);
  End
end;

////  Узнать входит ли элемент в дерево  \\\\
Procedure SearchInTree(root:^tree; poezd:integer);
Var found:boolean; i:integer;
Begin
  Found:=false;
  While (root<>nil) and (not found) and (i<>1000) do
  begin
    i:=i+1;
    if root^.data.n=poezd then
      found:=true
    else
    if root^.data.n>poezd then
      root:=root^.left
    else
      root:=root^.right;
  end;
```

```
  if found=false then
  begin
    Writeln('Я не нашёл такой поезд :(');
    readkey;
    exit;
  end
  else
  begin
    writeln('Поезд найден!!! :)');
    Writeln('-----------------------------------------------------');
    Writeln('|    Номер поезда:       | ',root^.data.n:25,'|');
    Writeln('|  Станция отправления: | ',root^.data.sO:25,'|');
    Writeln('|  Станция прибытия:    | ',root^.data.sP:25,'|');
    Writeln('|  Время отправления:   | ',root^.data.vO:25,'|');
    Writeln('|   Время прибытия:     | ',root^.data.vP:25,'|');
    Writeln('|  Стоимость билета:    | ',root^.data.cost:25,'|');
    Writeln('-----------------------------------------------------');
    Writeln;
    readkey;
  end;
end;

////  Сохранение одного элемента  \\\\
Procedure SaveElement(const elem:^tree; var fileP: file of rasp);
begin
  if elem = nil then exit;
  write(fileP, elem^.data);
  SaveElement(elem^.left, fileP);
  SaveElement(elem^.right, fileP);
end;

////  Сохранения дерева в файл  \\\\
Procedure SaveTree(const root:^tree; FileName:string);
var fileP: file of rasp;
begin
  assign(fileP, fileName);
  rewrite(fileP);

  SaveElement(root,fileP);

  close(fileP);
  writeln;
end;

////  Загрузка дерева из файла  \\\\
Procedure DownloadTree(var root:^tree; const FileName:string);
var fileP: file of rasp;
    element:^tree;
begin
  assign(fileP, fileName);
  reset(fileP);
  DeleteTree(root);
  while not eof(fileP) do
  begin
    new(element);
    element^.right:=nil;
    element^.left:=nil;
```

```
    read(fileP, element^.data);
    AddElement(root,element);
    element:=nil;
  end;
  close(fileP);
  writeln;
end;

Var FileName:string;
    num,otstup:integer;
    myTree:^tree;

////  Основная программа  \\\\
BEGIN
  FileName:='C:\PABCWork.NET\lab_9.txt';

  Writeln;
  Writeln('=============================================');
  Writeln('  Программа работает с бинарными деревьями  ');
  Writeln('  (дерево упорядочено по номерам поездов)  ');
  Writeln('=============================================');
  Writeln;
  readkey;
  clrscr;

  While true do
  begin
    Writeln;
    Writeln('1 - Создание нового дерева');
    Writeln('2 - Удаление дерева');
    Writeln('3 - Просмотр дерева');
    Writeln('4 - Узнать входит ли элемент в дерево');  //my
    Writeln('5 - Сохранить дерево');
    Writeln('6 - Загрузить дерево');
    Writeln('7 - Выход из программы');
    Writeln;
    Write('Ввод -> ');readln(num);
    ClrScr;
  case num of
    1:
    Begin
      CreateTree(myTree);
      clrscr;
    End;
    2:
    Begin
      Writeln('Дерево было удалено');
      DeleteTree(myTree);
      readkey;
      clrscr;
    End;
    3:
    Begin
      Writeln('Выполняется просмотр дерева':50);
      Writeln('---------------------------':50);
      Writeln('|      Номер поезда       |':50);
      Writeln('|   Станция отправления   |':50);
```

```
      Writeln('|    Станция прибытия      |':50);
      Writeln('|    Время отправления     |':50);
      Writeln('|     Время прибытия       |':50);
      Writeln('|    Стоимость билета      |':50);
      Writeln('----------------------------':50);
      Writeln;

      ShowTree(myTree,otstup);

      readkey;
      ClrScr;
    end;
    4:
    Begin
      Writeln('Выполняется проверка присутствия элемента в дереве');
      Write('Введите номер искомого поезда -> '); readln(num);
      SearchInTree(myTree,num);
      ClrScr;
    end;
    5:
    Begin
      SaveTree(myTree,FileName);
      Writeln('Дерво было успешно сохранено!');
      readkey;
      ClrScr;
    end;
    6:
    Begin
      DownloadTree(myTree,FileName);
      Writeln('Дерво было успешно загружено!');
      readkey;
      ClrScr;
    end;
    7: Exit;
  end;
  end;
END.
```

9.4.4 Выполнена отладка программы. Результаты тестирования отображены на рисунках 9.4, 9.5, 9.6, 9.7. На рисунках 9.4 и 9.5 видно, как запускается программа, которая предлагает выбрать одну из семи процедур и мы как сознательные пользователи выбираем первую процедуру и начинаем создавать дерево.

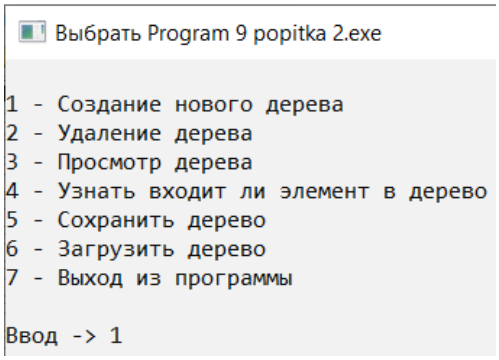

Рисунок 9.4 – Главное меню программы

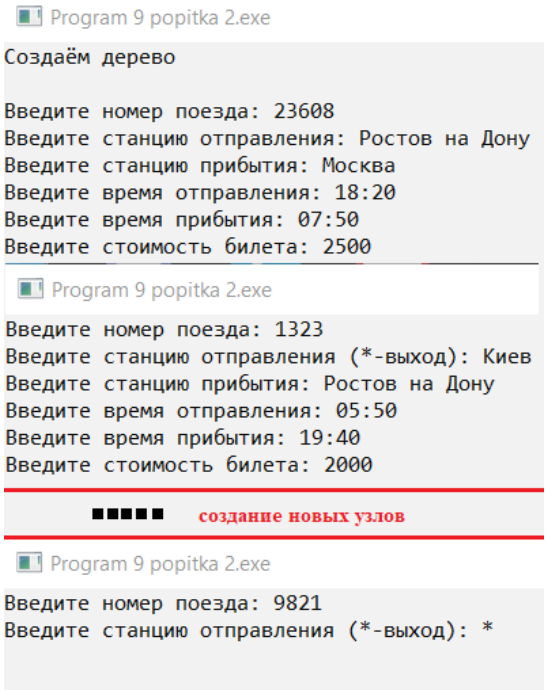

Рисунок 9.5 – Создание дерева

На рисунке 9.6 мы вызываем процедуру демонстрации дерева.

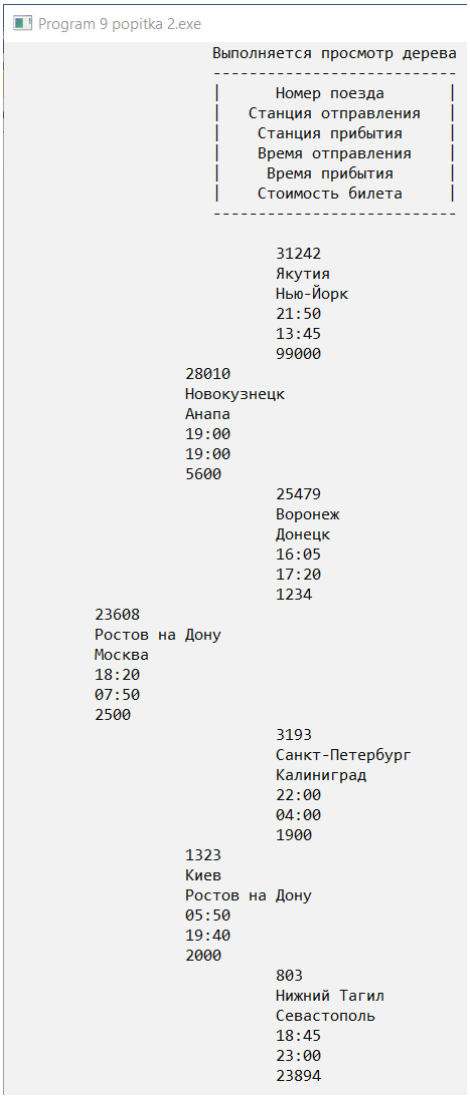

Рисунок 9.6 – Процедура демонстрации дерева

На рисунке 9.7 мы вызываем четвёртую процедуру которая проверяет входит ли элемент в дерево. В первый раз мы вводим несуществующий поезд, а во второй раз существующий под номером «25479».

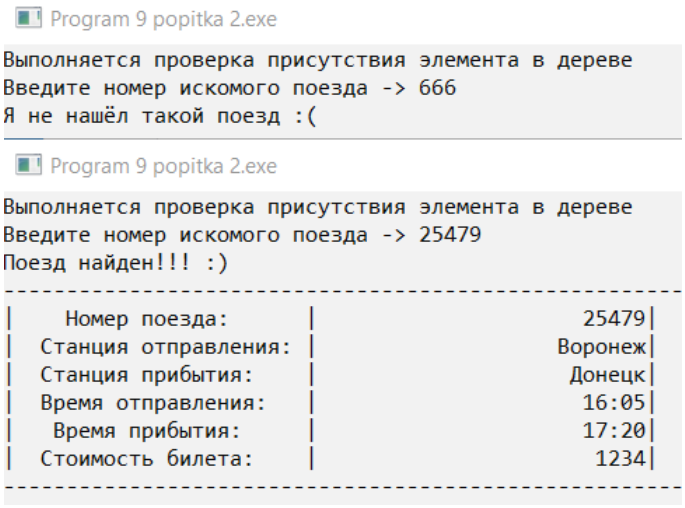

Рисунок 9.7 – Процедура проверки наличия элемента в дереве

На рисунках 9.8 и 9.10 мы сохраняем дерево в файл, затем удаляем наше дерево второй процедурой, в результате чего нашего дерева в программе нет. Затем с помощью процедуры загрузки мы загружаем дерево из файла в программу.

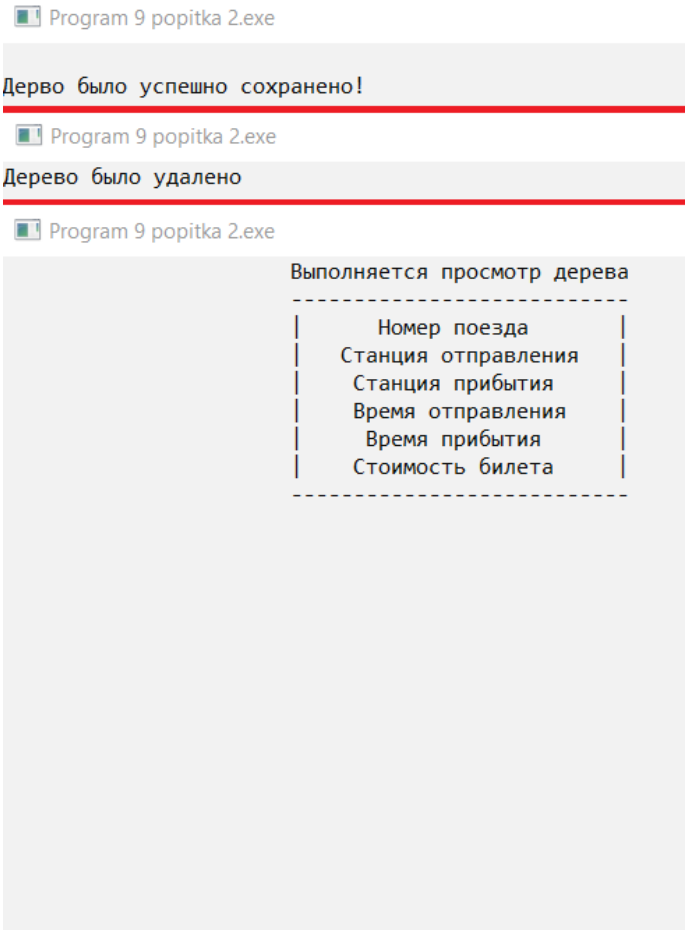

Рисунок 9.8 – Процедуры сохранения, удаления и демонстрации дерева

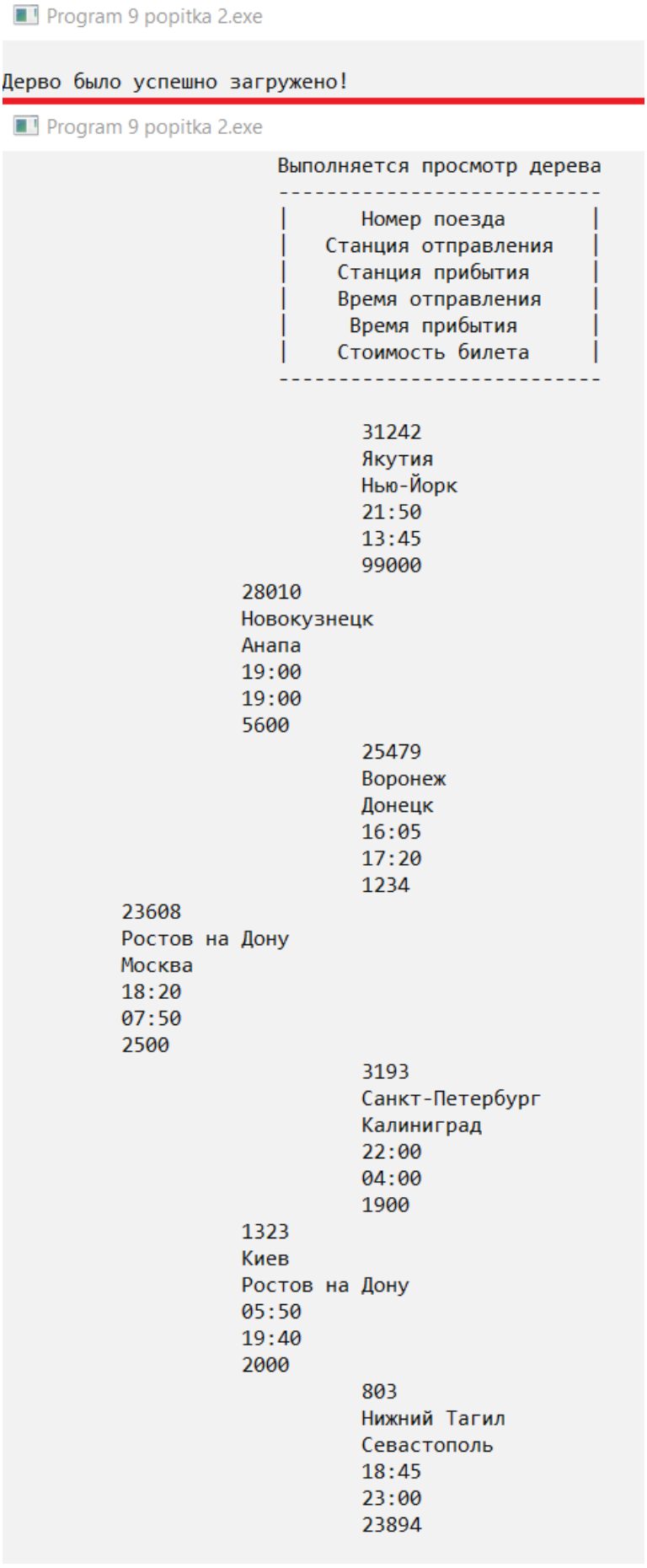

Рисунок 9.10 – Процедуры загрузки и демонстрации дерева

Результаты тестирования полностью соответствуют ожиданиям.

Выводы

В ходе выполнения данной лабораторной работы были получены навыки разработки программ, умеющих работать с бинарными деревьями; написаны процедуры и функции, осуществляющие главные операции над деревьями, а именно: создание дерева, удаление дерева, добавление элементов и различные мелкие процедуры. Также были повторно закреплены навыки отладки программы, работы с файлами, а именно чтение данных из бинарного типизированного файла и запись в него. Полученные навыки в будущем помогут создавать более сложные структуры деревьев, использовать деревья для упорядоченного хранения данных и эффективного поиска в них.