Министерство образования Республики Беларусь
Учреждение образования
“Белорусский государственный университет
информатики и радиоэлектроники”

Кафедра “Вычислительные методы и программирование”

А.К. Синицын, А.А. Навроцкий, А.В. Щербаков

ПРОГРАММИРОВАНИЕ АЛГОРИТМОВ В СРЕДЕ BUILDER C++

Лабораторный практикум по курсам «Программирование»
и «Основы алгоритмизации и программирование»
для студентов 1 – 2-го курсов всех специальностей БГУИР
дневной и вечерней форм обучения
В 2-х частях

Часть 2

Под общей редакцией А. К. Синицына

Минск 2005

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

УДК 681.3.06 (075.8)
ББК 32.973 я 73
С 38

Синицын А.К.

С 38 Программирование алгоритмов в среде Builder C++: Лаб. практикум по курсам «Программирование» и «Основы алгоритмизации и программирование» для студ. 1 – 2-го курсов всех спец. БГУИР дневной и вечерней форм обуч.: В 2 ч. Ч. 2 / А.К. Синицын, А.А. Навроцкий, А.В. Щербаков; Под общ. ред. А.К. Синицына. – Мн.: БГУИР, 2005. – 80 с.: ил.

ISBN 985-444-783-9 (ч. 2)

Лабораторный практикум содержит 10 тем, в которых рассмотрены краткие теоретические сведения об алгоритмах решения вычислительных задач, а также примеры их использования на языке C++в среде Builder. После каждой темы приведен набор индивидуальных заданий.

УДК 681.3.06 (075.8)
ББК 32.973 я 73

Часть 1 издана в БГУИР в 2004 г. Программирование алгоритмов в среде BUILDER C++. Лаб. практикум по курсам «Программирование» и «Основы алгоритмизации и программирование» для студ. 1-2-го курсов всех спец. БГУИР дневн. и веч. форм обуч. В 2 ч. Ч. 1 / А.К. Синицын, А.А. Навроцкий, А.В. Щербаков и др. – Мн.: БГУИР, 2004. – 92 с.: ил.

ISBN 985-444-783-9 (ч. 2)
ISBN 985-444-583-6

© Синицын А.К.,. Навроцкий А.А, Щербаков А.В., 2005
© БГУИР, 2005

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

СОДЕРЖАНИЕ

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

ТЕМА 1. ПРОГРАММИРОВАНИЕ С ОТОБРАЖЕНИЕМ ГРАФИЧЕСКОЙ ИНФОРМАЦИИ

Цель лабораторной работы: изучить возможности построения графиков с помощью компонента TChart. Научиться работать с графическими объектами. Написать и отладить программу с использованием функций отображения графической информации.

1.1. Как строится график с помощью компонента TChart

Обычно результаты расчетов представляются в виде графиков и диаграмм. Система BUILDER имеет мощный пакет стандартных программ вывода на экран и редактирования графической информации, который реализуется с помощью визуально отображаемого на форме компонента TChart.

Построение графика (диаграммы) производится после вычисления таблицы значений функции y=f(x). Полученная таблица с помощью метода AddXY передается в специальный двухмерный массив Chart**i**.Series[**k**] компонента TChart (**k** – номер графика (0,1,2,...)). Компонент TChart осуществляет всю работу по отображению графиков: строит и размечает оси, рисует координатную сетку, подписывает название осей и самого графика, отображает переданную таблицу в виде всевозможных графиков или диаграмм.

1.2. Использование класса TCanvas

Для рисования в BUILDER используется класс TCanvas, который является не самостоятельным компонентом, а свойством многих компонентов, и представляет собой холст (контекст GDI в Windows) с набором инструментов для рисования. Каждая точка холста имеет свои координаты. Начало осей координат располагается в верхнем левом углу холста. Данные по оси X увеличиваются слева направо, а по оси Y сверху вниз.

Основные свойства класса TCanvas:

__property TPen* Pen – карандаш (определяет параметры линий);

__property TBrush* Brush – кисть (определяет фон и заполнение замкнутых фигур);

__property TFont* Font – шрифт (определяет параметры шрифта).

Некоторые методы класса TCanvas:

void __fastcall Ellipse(int X1, int Y1, int X2, int Y2) – чертит эллипс в охватывающем прямоугольнике (X1, Y1), (X2, Y2). Заполняет внутреннее пространство эллипса текущей кистью;

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

void __fastcall LineTo(int X, int Y) – чертит линию от текущего положения пера до точки (X. Y);

void __fastcall MoveTo(int X, int Y) – перемещает карандаш в положение (X, Y) без вычерчивания линий;

void __fastcall Polygon(const TPoint * Points, const int Points_Size) – вычерчивает карандашом многоугольник по точкам, заданным в массиве Points. Конечная точка соединяется с начальной и многоугольник заполняется кистью. Для вычерчивания без заполнения используйте метод Polyline;

void __fastcall Rectangle(int X1, int Y1, int X2, int Y2) – вычерчивает и заполняет прямоугольник (X1, Y1), (X2, Y2). Для вычерчивания без заполнения используйте FrameRect или Polyline;

void __fastcall TextOut(int X, int Y, const AnsiString Text) – выводит текстовую строку Text так, чтобы левый верхний угол прямоугольника, охватывающего текст, располагался в точке (X, Y).

1.3. Пример написания программы

Задание: Составить программу, отображающую движение автомобиля с изменяющейся скоростью. Вывести с помощью компонента TChar график скорости.

1.3.1. Настройка формы

Панель диалога программы организуется в виде, представленном на рис.1.1. Причем поле с автомобилем не видно на этапе настройки формы и появляется

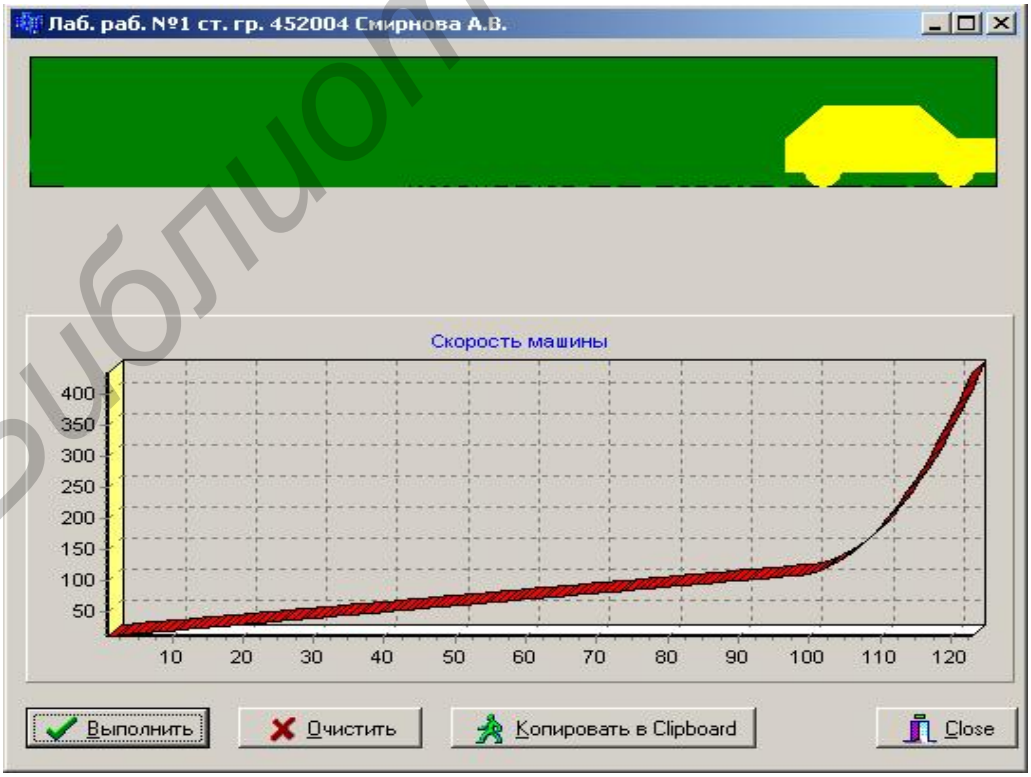


Рис. 1.1

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

только как результат выполнения кода программы. Поэтому в том месте формы, где предполагается делать вывод графической информации, следует оставить свободное место.

Компонент TChart вводится в форму путем нажатия пиктограммы в меню компонентов.

1.3.2. Работа с компонентом TChart

Для изменения параметров компонента TChart необходимо дважды щелкнуть по нему мышью в окне формы. Появится окно редактирования EditingChat1 (рис. 1.2). Для создания нового объекта Series1 нажать кнопку Add на странице

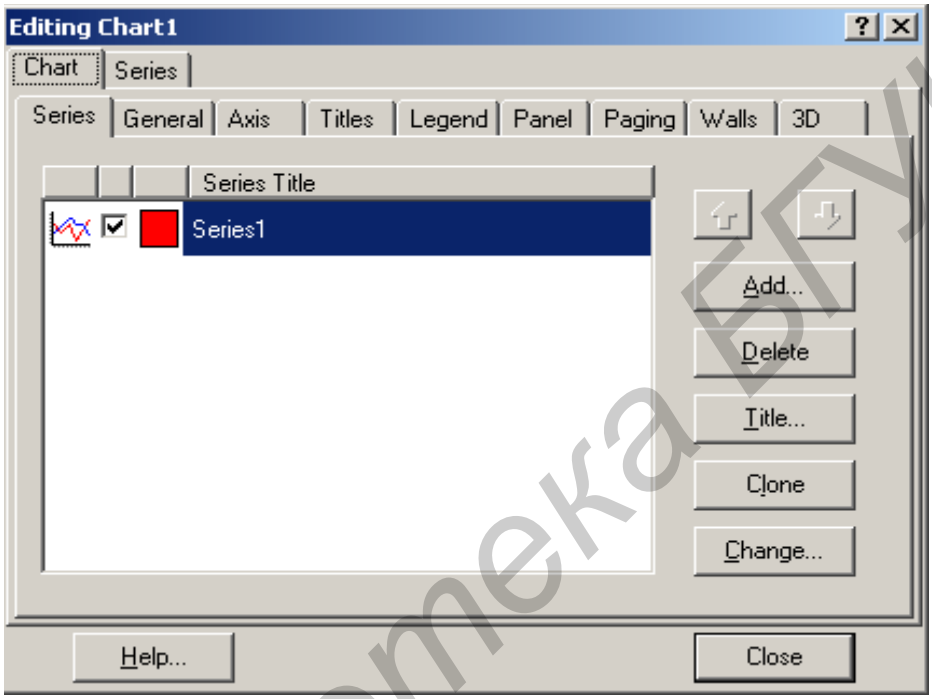


Рис. 1.2

Series. В появившемся диалоговом окне TeeChart Gallery выбрать пиктограмму с надписью Line (график выводится в виде линий). Если нет необходимости представления графика в трехмерном виде, отключается независимый переключатель 3D. После нажатия на кнопку OK появится новая серия с название Series1. Для изменения названия нажать кнопку Title. Название графика вводится на странице Titles. Разметка осей меняется на странице Axis.

Данные по оси X автоматически сортируются, поэтому, если необходимо нарисовать, например, окружность, сортировку отключают функцией Order : Chart1->Series[0]->XValues->Order=loNone.

Нажимая различные кнопки меню, познакомьтесь с другими возможностями редактора EditingChat.

Текст программы имеет вид:

```
#include <vcl.h>
#pragma hdrstop
#include "lr01.h"
```

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

```
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------

void Shar(int xx,int yy, TColor cc)  // Вывод изображения машины
{
  Form1->Canvas->Pen->Color=cc;   // Установка цвета карандаша
  Form1->Canvas->Brush->Color=cc;  // Установка цвета кисти
        TPoint pnt[7];
    pnt[0]=Point(xx+0,yy+20);
    pnt[1]=Point(xx+0,yy+40);
    pnt[2]=Point(xx+110,yy+40);
    pnt[3]=Point(xx+110,yy+20);
    pnt[4]=Point(xx+90,yy+20);
    pnt[5]=Point(xx+70,yy+0);
    pnt[6]=Point(xx+20,yy+0);
 Form1->Canvas->Polygon(pnt,6);  // Кузов машины
 Form1->Canvas->Ellipse(xx+10,yy+30,xx+30,yy+50);  // Колесо
 Form1->Canvas->Ellipse(xx+80,yy+30,xx+100,yy+50); // Колесо
}   // Конец функции вывода изображения машины
//---------------------------------------------------------------------------

void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{
  Canvas->Pen->Color = clBlack;  // Установка цвета карандаша
  Canvas->Brush->Color = clGreen;       // Установка цвета кисти
    Canvas->Rectangle(10,10,520,90);  // Рисует прямоугольник
   double hx=1;  double h=0;
   double x=10;  int n=0;
 while (x<410) {
    h+=0.01;
    hx+=int (h);             // Изменения шага для увеличения скорости
  Shar(x,40,clYellow);           // Рисование машины
     Sleep(10);                  // Задержка
  Shar(x,40,clGreen);            // Стирание нарисованной машины
     x+=hx;
      n++;
```

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

```
   Chart1->Series[0]->AddXY(n,x,"",clTeeColor); // Ввод данных для графика
                }
}
//---------------------------------------------------------------------------
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
 Chart1->Series[0]->Clear();  // Очистка графика
}
//---------------------------------------------------------------------------
void __fastcall TForm1::BitBtn3Click(TObject *Sender)
{
 Chart1->CopyToClipboardMetafile(True);// Передача графика в буфер
                                       // обмена для последующей вставки в отчет
}
//---------------------------------------------------------------------------
```

1.4. Выполнение индивидуального задания

Решить задачу в соответствии с заданным вариантом и, используя компонент Tchar, нарисовать соответствующие геометрические фигуры.

1. Даны три числа *a, b, c*. Необходимо определить, существует ли треугольник с такими длинами сторон.

2. Даны четыре числа *a, b, c, d*. Необходимо определить, существует ли четырехугольник с такими длинами сторон.

3. Найти взаимное расположение двух окружностей радиусами R_1 и R_2 с центрами в точках (x_1, y_1), (x_2, y_2) соответственно.

4. Найти взаимное расположение окружности радиусом R с центром в точке (x_0, y_0) и прямой, проходящей через точки с координатами (x_1, y_1) и (x_2, y_2) (пересекаются, касаются, не пересекаются).

5. Определить количество точек с целочисленными координатами, лежащих внутри окружности радиусом R с центром в точке (x_0, y_0).

6. Найти координаты точек пересечения двух окружностей радиусами R_1 и R_2 с центрами в точках (x_1, y_1) и (x_2, y_2) соответственно.

7. Найти координаты точки, симметричной данной точке M с координатами (x_1, y_1), относительно прямой $Ax+By+C=0$.

8. Даны две точки M_1 (x_1, y_1), $M_2(x_2, y_2)$ и прямая $Ax+By+C=0$. Необходимо найти на этой прямой такую точку M_0 (x_0, y_0), чтобы суммарное расстояние от нее до двух данных точек было минимально.

9. Даны три точки с координатами (x_1, y_1), (x_2, y_2), (x_3, y_3), которые являются вершинами некоторого прямоугольника со сторонами, параллельными осям координат. Найти координаты четвертой точки.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

10. Даны координаты четырех точек (x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4). Необходимо определить, образуют ли они выпуклый четырехугольник.

11. Даны координаты четырех точек (x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4). Необходимо определить, образуют ли они: а) ромб; б) квадрат; в) трапецию.

12. Даны координаты двух вершин (x_1, y_1) и (x_2, y_2) некоторого квадрата. Необходимо найти возможные координаты других его вершин.

13. Даны координаты двух вершин (x_1, y_1) *и* (x_2, y_2) некоторого квадрата, которые расположены по диагонали, и точка (x_3, y_3). Необходимо определить, лежит или не лежит точка внутри квадрата.

14. Даны координаты трех вершин (x_1, y_1), (x_2, y_2), (x_3, y_3) треугольника. Необходимо найти координаты точки пересечения его медиан.

15. Даны координаты трех вершин (x_1, y_1), (x_2, y_2), (x_3, y_3) треугольника. Необходимо найти длины его высот.

16 – 30. Постройте графики функций для соответствующих вариантов из темы №1. Таблицу данных получить, изменяя параметр X с шагом h. Ввод исходных данных организовать через окна TEdit. Самостоятельно выбрать удобные параметры настройки.

31. Разработать программу, реализующую игру «Бега лошадей по кругу ипподрома». Предусмотреть возможность устанавливать ставки на лошадей и рассчитывать выигрыш. Скорость движения лошадей должна задаваться случайным образом функцией Random.

32. Разработать программу, реализующую игру «Бега лошадей по прямой». Предусмотреть возможность устанавливать ставки на лошадей и рассчитывать выигрыш. Скорость движения лошадей должна задаваться случайным образом функцией Random.

33. Разработать программу игры в крестики – нолики. В основу положить компонент DrawGrid.

34. Разработать программу игры «Минер» по подобию такой же игры в системе Windows. Начальная расстановка мин должна выполняться случайным образом. В основу положить компонент DrawGrid.

35. Разработать программу игры стрельбы из подводной лодки по кораблю, используя вид из перископа. На заднем плане должен периодически проплывать кораблик с постоянной поперечной скоростью. С помощью клавиш “влево” “вправо” следует менять вид в перископе. Клавиша «Ввод» должна запускать торпеду. В перископе должна отображаться траектория движения торпеды с уменьшением скорости движения при приближении к кораблю. Попадание должно сопровождаться видимым взрывом и исчезновением корабля.

36. Разработать программы игры «Бомбометание с самолета по наземной цели». С летящего с постоянной скоростью самолета клавишей «Ввод» производить бомбометание. Траектория движения бомбы должна соответствовать физи-

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

ческим законам падения тел на землю. Попадание в цель должно сопровождаться видимым взрывом и исчезновением цели. Самолет должен периодически вылетать из-за края канвы компонента рисования.

37. Разработать программу игры «Морской бой». Программа должна случайным образом на сетке 10x10 расставлять корабли: один четырехклеточный, два трехклеточных, три двухклеточных и четыре одноклеточных. Они не могут изгибаться и соприкасаться друг с другом. Игрок выбирает определенный квадрат и как бы стреляет в него. Программа должна сообщать, попал ли игрок в корабль или нет. Она также должна отображать все старые выстрелы и показывать ячейки, куда уже не имеет смысла стрелять. Аналогично строится и вторая таблица, где игрок располагает свои корабли, по которым уже случайным образом стреляет программа. Выигрывает тот, кто быстрей потопит корабли неприятеля. Предусмотреть в конце игры показ расположения кораблей программы. Для таблиц использовать компоненты TdrawGrid.

38. Разработать программы игры «Стрельба из пушки». Пушка должна стрелять через гору по какой-то цели. Траектория полета снаряда должна подчиняться законам физики. Игрок может управлять углом подъема ствола относительно горизонта и начальной скоростью снаряда в дискретных величинах (определяется типом снаряда). При попадании должен происходить видимый взрыв и исчезновение цели.

39. Разработать программу показа в форме текущего времени в виде обычных стрелочных часов со стрелками часов, минут и секунд.

40. Разработать программу простейшего графического редактора (аналога программы Paint системы Windows). Он должен рисовать в канве компонента TpaintBox произвольные кривые с помощью мыши. Предусмотреть возможность:
а) изменения толщины кривых;
б) изменения цвета кривых;
в) сохранения рисунка в графическом файле.

41. Разработать программу простейшего графического редактора (аналога программы Paint системы Windows). Он должен рисовать в канве компонента TpaintBox ломаные линии с помощью нажатия на клавиши мыши. Предусмотреть возможность:
а) изменения толщины линий;
б) изменения цвета линий;
в) сохранения рисунка в графическом файле.

42. Разработать программу простейшего графического редактора (аналога программы Paint системы Windows). Он должен рисовать в канве компонента TpaintBox с помощью мыши прямоугольники. Предусмотреть возможность:
а) изменения толщины линий;
б) изменения цвета линий;
в) заливки областей текущей кистью;
г) изменения цвета кисти;
д) сохранения рисунка в графическом файле.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

43. Разработать программу простейшего графического редактора (аналога программы Paint системы Windows). Он должен рисовать в канве компонента TpaintBox с помощью мыши эллипсов. Предусмотреть возможность:
а) изменения толщины линий;
б) изменения цвета линий;
в) заливки областей текущей кистью;
г) изменения цвета кисти;
д) сохранения рисунка в графическом файле.

44. Разработать программу простейшего графического редактора (аналога программы Paint системы Windows). Он должен рисовать в канве компонента TpaintBox любой текст в указанном мышью месте. Предусмотреть возможность:
а) изменения типа, размера и цвета шрифта;
б) сохранения рисунка в графическом файле.

45. Разработать программу простейшего графического редактора (аналога программы Paint системы Windows). Он должен помещать в канву компонента TpaintBox из графического файла произвольный рисунок и обеспечивать возможность:
а) стирания произвольной области рисунка;
б) изменения размеров стирки;
в) сохранения рисунка в графическом файле.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

ТЕМА 2. ПРОГРАММИРОВАНИЕ С ИСПОЛЬЗОВАНИЕМ РЕКУРСИИ

Цель лабораторной работы: изучить способы программирования алгоритмов с использованием рекурсии.

2.1. Понятие рекурсии

Рекурсия – это такой способ организации вычислительного процесса, при котором подпрограмма в ходе выполнения составляющих ее операторов обращается сама к себе. Классический пример программирования вычисления *n*-го члена ($n>0$) рекуррентной последовательности $x_n = j\ (x_{n-1})$ при $x_0 = a$:

```
double XR(int n)
{
      if (n==0) return a;              // Условие окончания рекурсии
          else return φ(XR(n-1));      // Рекурсивное обращение к функции
 }
```

Здесь φ - функция, определяющая закон рекуррентности.

При выполнении правильно организованной рекурсивной подпрограммы осуществляется многократный переход от некоторого текущего уровня организации алгоритма к нижнему уровню последовательно, до тех пор пока наконец не будет получено тривиальное решение задачи (в вышеприведенном примере $n=0$).

Рекурсивная форма записи алгоритма обычно выглядит изящнее итерационной и дает более компактный текст программы, но при выполнении, как правило, медленнее и может вызвать переполнение программного стека. Это вызвано тем, что при активации каждого экземпляра вложенной подпрограммы порождается новое множество локальных переменных и, кроме того, запоминается текущее состояние вычислений. Поэтому рекурсивные алгоритмы обычно используются в тех случаях, когда в структуре решаемой задачи рекурсия присутствует явно.

Рекурсивный вызов может быть прямым, как в вышеприведенном примере, и косвенным. В этом случае подпрограмма обращается к себе опосредованно, путем вызова другой подпрограммы, в которой содержится обращение к первой. Косвенный вызов может быть явно не виден в тексте программы, поэтому его использование без необходимости нежелательно.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

2.2. Порядок выполнения работы

Задание: Написать программу, содержащую две подпрограммы, вычисляющие факториал числа n. Одна вычисляет факториал без использования рекурсии, другая - рекурсивная.

2.2.1. Пример рекурсивной и нерекурсивной подпрограммы

Подпрограммы имеют следующий вид:

```
int factorialR(int n)      // Программа, использующая рекурсию
{
    if (n==0) return 1;
        else return n*factorialR(n-1);
}

int factorial(int n)       // Программа, не использующая рекурсию
{
    int f=1;
    for (int i=1; i<=n; i++) f=f*i;
        return f;
}
```

2.3. Варианты задач

Решить поставленные задачи двумя способами – с применением рекурсии и без нее.

1. Для заданного целого десятичного числа N получить его представление в p-ичной системе счисления (p<10).

2. В упорядоченном массиве целых чисел a_i, $i=1...n$ найти номер элемента c, используя метод двоичного поиска. Предполагается, что элемент c находится в массиве.

3. Найти наибольший общий делитель чисел M и N. Используйте теорему Эйлера: если M делится на N, то $НОД\ (N, M)=N$, иначе $НОД\ (N, M)= НОД\ (M \bmod N, N)$.

4. Вычислить число Фибоначчи $Fb(n)$. Числа Фибоначчи определяются следующим образом: $Fb(0)=0;\ Fb(1)=1;\ Fb(n)=Fb(n-1)+Fb(n-2)$.

5. Найти значение функции Аккермана $A(m, n)$, которая определяется для всех неотрицательных целых аргументов m и n следующим образом:

$A(o, n)=n+1;$

$A(m, o)=A(m-1, 1);\ (m>o);$

$A(m, n)=A(m-1, A(m, n-1));\ (m>o;\ n>o).$

6. Найти методом деления отрезка пополам минимум функции $f(x)=x-7\sin^2(x)$ на интервале $[2, 6]$ с точностью ε=0,1.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

7. Вычислить значение $x=\sqrt{a}$, используя рекуррентную формулу $x_n=\frac{1}{2}\left(x_{n-1}+a/x_{n-1}\right)$, в качестве начального приближения использовать значение $x_0=0.5(1+a)$.

8. Найти максимальный элемент в массиве $a_1...a_n$, используя очевидное соотношение $max\ (a_1...a_n)=max\ (max\ (a_1...a_{n-1}),\ a_n)$.

9. Найти максимальный элемент в массиве $a_1...a_n$, используя соотношение (метод деления пополам) $max\ (a_1...a_n)=max\ (max\ (a_1...a_{n/2}),\ max\ (a_{n/2+1},\ a_n))$.

10. Вычислить $y(n)=\sqrt{1+\sqrt{2+...+\sqrt{n}}}$.

11. Вычислить $y(n)=\cfrac{1}{n+\cfrac{1}{(n-1)+\cfrac{1}{(n-2)+\cfrac{1}{.. \quad ..}}}}$

$$....+\cfrac{1}{1+\frac{1}{2}}.$$

12. Вычислить произведение n≥2 (n-четное) сомножителей $y=\frac{2}{1}\cdot\frac{2}{3}\cdot\frac{4}{3}\cdot\frac{4}{5}\cdot\frac{6}{5}\cdot\frac{6}{7}\cdot...$.

13. Вычислить $y=x^N$ по следующему алгоритму: $y=(x^{N/2})^2$, если N четное; $y=x\cdot y^{N-1}$, если N нечетное.

14. Дан массив чисел. Все числа, граничащие с цифрой "1", заменить нулями.

15. Игра "Ханойские башни" состоит в следующем. Есть три стержня. На первый из них надета пирамида из ***n*** колец (большие кольца снизу, меньшие сверху). Требуется переместить кольца на другой стержень. Разрешается перекладывать кольца со стержня на стержень, но класть большее кольцо поверх меньшего нельзя. Составить программу, указывающую требуемые действия.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

ТЕМА 3. ПРОГРАММИРОВАНИЕ С ИСПОЛЬЗОВАНИЕМ ДЕРЕВЬЕВ РЕШЕНИЙ

Цель лабораторной работы: изучить способы программирования алгоритмов с использованием деревьев решений.

3.1. Задача оптимального выбора и дерево решений

Много важных прикладных задач (в частности задач искусственного интеллекта) можно свести к следующей:

Имеется набор из n элементов $a_1 ... a_n$. Каждый элемент a_i характеризуется определенными свойствами, например, вес w_i и цена c_i (это могут быть размер и время, объем инвестиций и ожидаемый доход и т.д.). Требуется найти оптимальную выборку $a_{i_1} ... a_{i_k}$ из этого набора, т.е. такую, для которой, например, при заданном ограничении на суммарный вес $\sum_{j=1}^{k} w_{i_j} \le W_{max}$ достигается максимальная стоимость $\sum_{j=1}^{k} c_{i_j}$.

С такой проблемой сталкивается, например, путешественник, упаковывающий чемодан, суммарный вес которого ограничен W_{max} кг, а ценность вещей должна быть больше, или инвестор, которому надо выгодно вложить W_{max} млн р. в какие-то из n возможных проектов, каждый из которых имеет свою стоимость и ожидаемый доход.

Решение данной задачи состоит в переборе всех возможных выборок $\{(i_1,...,i_k), 0 \le k \le n\}$ из n значений $\{1, 2, ..., n\}$. Например, для n=3 таких выборок $2^n = 8$: $\{1, 2, 3\}$, $\{1, 2\}$, $\{1, 3\}$, $\{2, 3\}$, $\{1\}$, $\{2\}$, $\{3\}$, $\{\ \}$. Наглядной моделью решения данной задачи служит двоичное дерево вида

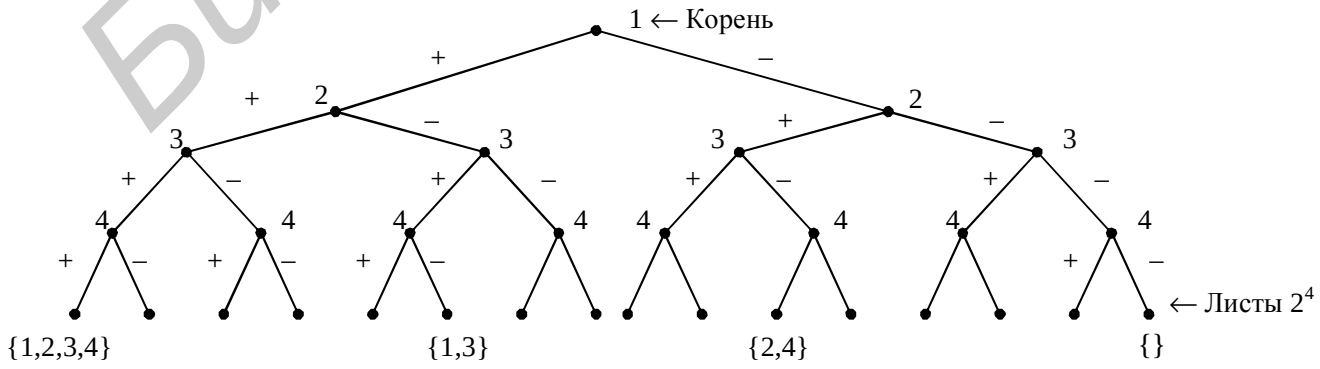


Рис. 3.1

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

Каждый узел в дереве представляет собой один шаг решения задачи перебора вариантов. Ветвь в дереве соответствует решению, которое ведет к более полному решению. В игровых деревьях, например, каждая ветвь соответствует ходу игрока. Листья представляют собой окончательные решения (варианты).

Цель состоит в том, чтобы из всех возможных вариантов (решений) найти наилучший путь от корня до листа при выполнении некоторых условий.

Естественно, что форма дерева и условия, налагаемые на решения, определяются конкретной задачей. Так, для задачи оптимального выбора подходит вышеприведенное двоичное дерево. Для задачи моделирования игры в крестики-нолики дерево получается с большим количеством ветвей, выходящих из узла:

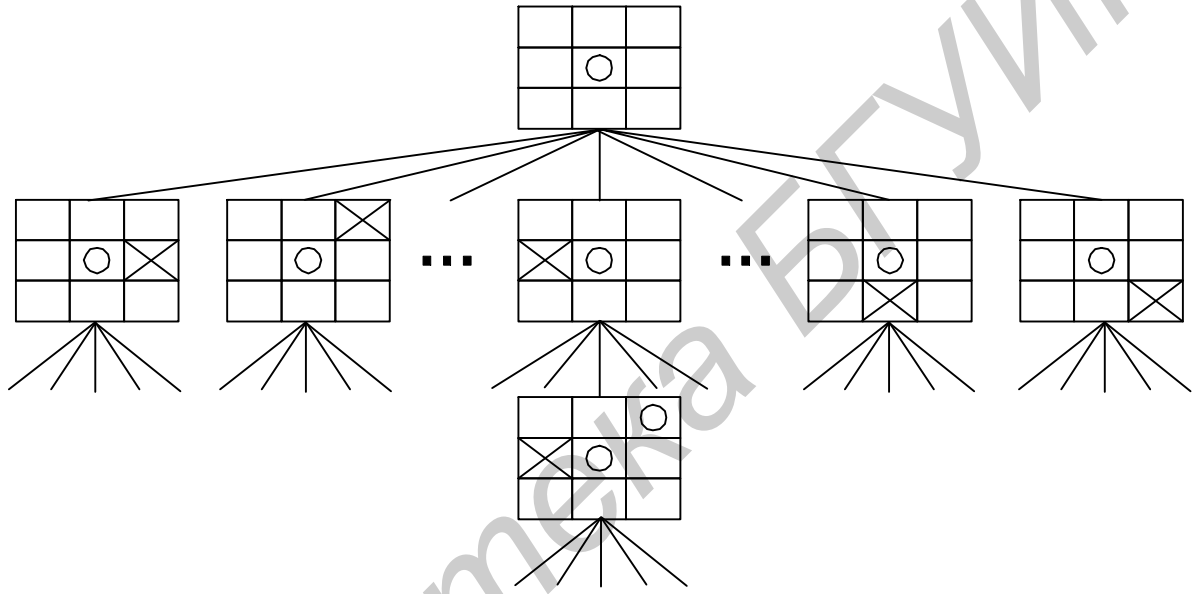


Рис. 3.2

В этой задаче выборки делаются из двухмерного исходного массива $\{1..n,\ j_i = 1..m_i\}$ - j-й кандидат на i-м ходу.

Деревья решений обычно огромны. Все дерево для игры в крестики-нолики содержит более 500 тыс. вариантов. Большинство же реальных задач несравненно сложнее. Соответствующие им деревья решений могут содержать больше листов, чем атомов во Вселенной.

3.2. Рекурсивные процедуры полного перебора игрового дерева

Имеется довольно много методов работы с такими огромными деревьями. Для самых маленьких деревьев (типа крестики-нолики) можно использовать **метод полного перебора** всех возможных решений.

Рекурсивная процедура полного перебора листьев произвольного дерева выглядит следующим образом:

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

```
 int Pbr(int i)  // Метод полного перебора
{
   do //  Начало перебора всех кандидатов
     {
  < выбор и включение в выборку j-го кандидата на i-м ходу >

   if(i<n-1) Pbr(i+1);
                  else
< лист решения сформирован, проверка приемлемости и оптимальности >;

< исключение j-го кандидата на i-м ходу из выборки >
     }
 while (< список кандидатов ai1 … aim_i  не исчерпан >);
}
```

Здесь в каждом узле дерева имеется набор *m* ветвей (кандидатов), по которым возможно продвижение к полному решению. Величина *m* может быть разная и определяется условием **<** список кандидатов ***ai1..aim$_i$*** не исчерпан**>.** Например, в ранее рассмотренной задаче на каждом ходу предлагается два кандидата {ai1=i, ai2=0}, в результате получим рекурсивный перебор двоичного дерева.

Полный перебор используется редко.

Для работы с большими деревьями более подходит рассмотренный ниже **метод ветвей и границ**. Идея этого метода в том, что на каждом шаге решения делается оценка целесообразности дальнейшего спуска по той или иной ветви и прекращение (отсечение) просмотра по пути заведомо не оптимальному. Такое отсечение позволяет значительно уменьшить количество допускаемых решений.

3.3. Рекурсивная процедура метода ветвей и границ

Стратегия отсечения заведомо неприемлемых и неоптимальных решений позволяет резко сократить количество просматриваемых вариантов. В общем случае для двоичного дерева эта стратегия программируется с помощью следующей рекурсивной процедуры:

```
int Vbr(int i, int ac)   // Метод ветвей и границ
{
 if (< приемлемо включение i-го элемента >)
   {
  <включение i-го элемента в выборку>;
 if(i<n-1) Vbr(i+1,ac);
       else <проверка оптимальности>;
  <исключение i-го элемента из выборки>;
          }
```

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

```
if (<приемлемо невключение i-го элемента >)
          {
 if(i<n-1) Vbr(i+1,ac1);
     else <проверка оптимальности>;
          }
  }
```

С помощью приведенной рекурсивной процедуры описывается процесс исследования на пригодность *i*-го элемента к включению в выборку и генерацию всех выборок с проверкой на оптимальность. При рассмотрении каждого элемента (кандидата на включение) возможны два заключения: включать или не включать элемент в текущую выборку. Причем условия включения и невключения в общем случае разные.

Ниже приведен текст программы, реализующей решение сформулированной задачи методом ветвей и границ. Здесь введен массив элементов *a*, имеющих тип записи с полями *w* и *c*. При генерации текущей и оптимальной выборок используются множества S и optS из индексов, входящих в выборку элементов. В процедуре Vbr(i,ac) введен дополнительный параметр: ac – общая текущая стоимость выборки, т.е. стоимость, которую еще можно достичь, продолжая текущую выборку на данном пути. Критерием <приемлемо включение *i*-го элемента> является тот факт, что он подходит по весовым ограничениям, т.е. tw+a[i].w≤Wmax.

Если элемент не подходит по этому критерию, то попытки добавить еще один элемент в текущую выборку можно прекратить. Если речь идет об исключении, то критерием <приемлемости, невключения>, т.е. возможности продолжения построения текущей выборки, будет то, что после данного исключения общая стоимость текущей выборки ac будет не меньше полученной до этого стоимости maxC оптимальной выборки, находящейся в optS: ac-a[i].c>maxC.

Ведь если сумма меньше, то продолжение поиска (хотя и может дать некоторое решение) не приведет к оптимальному решению. Следовательно, дальнейший поиск на текущем пути бесполезен.

3.4. Эвристические методы

Мы уже видели, что для программирования поиска в дереве решений рекурсивные процедуры оказываются довольно эффективным средством.

Если, однако, дерево очень большое, например, если в задаче оптимального выбора N=100 дерево содержит более 10^{20} узлов, то даже современный компьютер методом ветвей и границ будет решать эту задачу более 1 млн лет.

Для огромных деревьев остается единственный способ – использовать эвристический метод. Найденный эвристическим методом вариант может оказаться не наилучшим из возможных, но зачастую близким к нему. Эвристические методы позволяют исследовать практически любое дерево.

Некоторые эвристические методы решения задачи оптимального выбора:

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

Метод максимальной стоимости:

При каждом включении нового элемента в выборку выбирается элемент, имеющий максимальную цену, до тех пор, пока суммарный вес менее W_{max}.

Решается задача очень просто, без всяких рекурсий, особенно если массив элементов вначале отсортировать по стоимости c.

Метод наименьшего веса:

Эта стратегия в некотором смысле противоположна предыдущей.

На каждом шаге выбирается элемент с минимальным весом. В этом случае мы сформируем выборку с максимальным количеством элементов.

Метод сбалансированной стоимости:

Эта эвристика состоит в том, что при включении очередного элемента в выборку сравнивается как стоимость, так и вес, а именно, элемент с самым большим отношением стоимости к весу.

Метод случайного поиска:

Используя датчик случайных чисел, алгоритм выбирает очередной элемент случайным образом. Поиск осуществляется несколько раз. Данный метод дает удивительно хорошие результаты.

Сравнение:

Различные эвристические методы ведут себя по-разному в различных задачах. Рекомендуется попробовать решить задачу всеми вам известными эвристическими методами и выбрать наилучшее из полученных решений.

Следует отметить, что условия оптимальности в каждой задаче задаются по-разному и не всегда так просто сформулировать условия отсечения перспективных ветвей в методе ветвей и границ. Более того, даже метод сбалансированной стоимости не всегда так просто реализовать.

3.5. Порядок написания программы

Задание: Составить программу решения задачи оптимального выбора методом полного перебора и методом ветвей и границ.

Текст программы приведен ниже.

```
#include <vcl.h>
#pragma hdrstop
#include "lr01.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
```

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

```
{
}
//---------------------------------------------------------------------------
 typedef struct
   {
    int wes;
    int cena;
   } TElem;

 typedef Set<int,0,9> TSet;

   TElem zap[10];
   TSet S, optS;
   int Wmax, maxC;
   int i,n;
   int tw;  // Суммарный вес текущей  выборки
   int ac;  // Стоимость текущей  выборки

//---------------------------------------------------------------------------

int Pbr(int i)  // Метод полного перебора
{
 S << i;
  tw+=zap[i].wes;
  ac+=zap[i].cena;
    if(i<n-1) Pbr(i+1);
        else
      if (ac>maxC && tw<=Wmax) { maxC=ac; optS=S; }
  S >> i;
   tw-=zap[i].wes;
   ac-=zap[i].cena;
     if(i<n-1) Pbr(i+1);
        else
      if (ac>maxC && tw<=Wmax) { maxC=ac; optS=S; }
}

//---------------------------------------------------------------------------
```

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

```
int Vbr(int i, int ac)   // Метод ветвей и границ
{
 if (tw+zap[i].wes<=Wmax)
                    {
     S << i;
      tw+=zap[i].wes;
     if(i<n-1) Vbr(i+1,ac);
          else
        if (ac>maxC) { maxC=ac; optS=S;}
      S >> i;
       tw-=zap[i].wes;
                    }

  int ac1=ac-zap[i].cena;
   if (ac1>maxC){
      if(i<n-1) Vbr(i+1,ac1);
             else { maxC=ac1; optS=S; }
              }
}

//---------------------------------------------------------------------------

void __fastcall TForm1::FormCreate(TObject *Sender)
{
 Edit1->Text="10";
 Edit2->Text="50";
 StringGrid1->Cells[1][0]="Вес";
 StringGrid1->Cells[2][0]="Цена";
  for (int i=1; i<=10; i++) StringGrid1->Cells[0][i]=IntToStr(i-1);
  for (int i=1; i<=10; i++){
                StringGrid1->Cells[1][i]=IntToStr(i*i);
                StringGrid1->Cells[2][i]=IntToStr(i+5);
                        }
 Memo1->Clear();
}
//---------------------------------------------------------------------------
```

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
 S.Clear(); optS.Clear(); maxC=0;
 Memo1->Clear();
  n=StrToInt(Edit1->Text);
  Wmax=StrToInt(Edit2->Text);
       int ass=0;
     for (int i=0; i<n; i++) {
       zap[i].wes=StrToInt(StringGrid1->Cells[1][i+1]);
       zap[i].cena=StrToInt(StringGrid1->Cells[2][i+1]);
        ass+=zap[i].cena;
                          }

 switch (RadioGroup1->ItemIndex)
 {
 case 0: {
    Memo1->Lines->Add("Метод полного перебора");
   tw=0; ac=0;
    Pbr(0);
     break;
         }

 case 1: {
    Memo1->Lines->Add("Метод ветвей и границ");
   tw=0;
    Vbr(0,ass);
     break;
         }
 }

 for (i=0; i<n; i++)  // Вывод результата
 if (optS.Contains(i)) Memo1->Lines->Add(IntToStr(i)+"  "+
  IntToStr(zap[i].wes)+"   "+IntToStr(zap[i].cena));
Memo1->Lines->Add("  Wmax = "+IntToStr(Wmax)+"   maxC =
"+IntToStr(maxC));
}
//---------------------------------------------------------------------------
```

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

Панель диалога будет иметь вид, показанный на рис. 3.3.

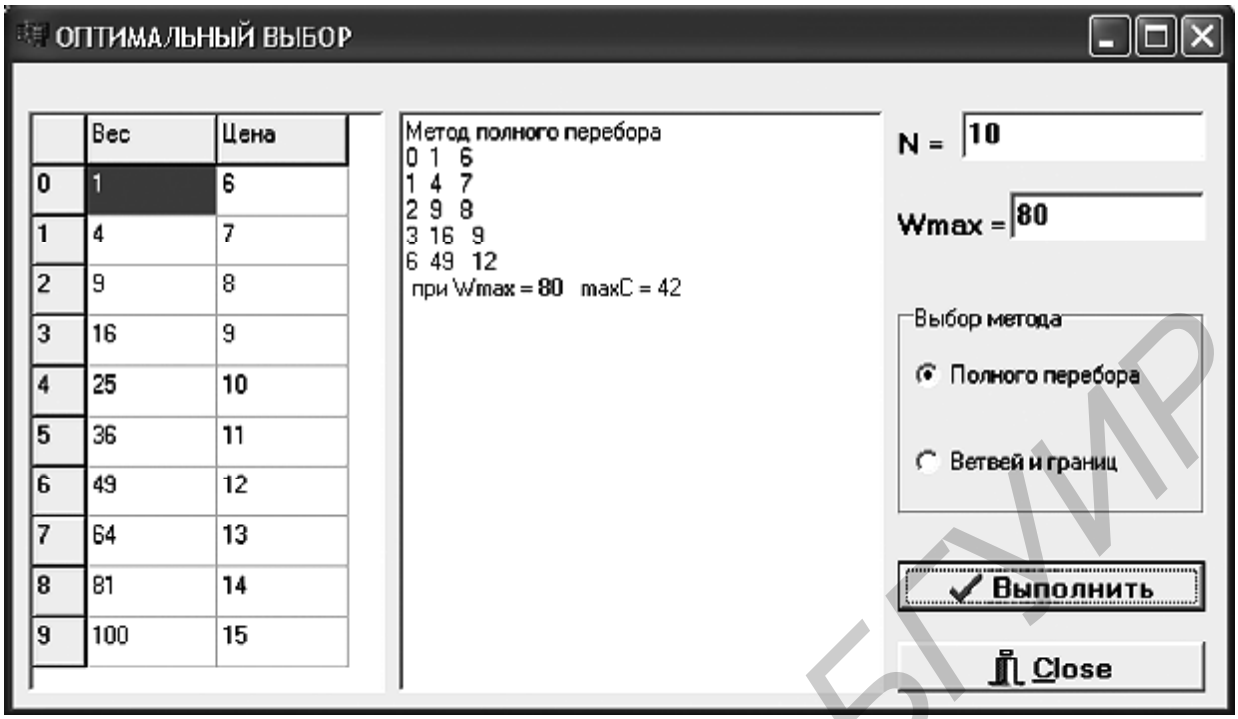


Рис. 3.3

3.6. Варианты задач

Задана таблица из 10 элементов:

Вес	$10+N_v$	11	12	13	14	15	16	17	18	19
Цена	18	20	17	19	$28-N_v$	21	27	23	25	24

Здесь N_v – номер варианта 1-15.

1. Решить задачу методом полного перебора и методом ветвей и границ. Для двух значений W_{max}=50 и W_{max}=10 найти оптимальные варианты и построить дерево поиска, поясняющее работу алгоритма, для чего в нужных местах вставить вывод промежуточных значений.

2. Решить эту же задачу методом полного перебора для n=5, построить полное дерево поиска, получить оценку эффективности метода ветвей и границ по отношению к методу полного перебора.

3. Решить задачу методом максимальной стоимости.

4. Решить задачу методом минимального веса.

5. Решить задачу методом сбалансированной стоимости.

6. Решить задачу методом случайного поиска (метод Монте-Карло).

Сравнить полученные решения и время выполнения для n=10, 20, 40.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

ТЕМА 4. ПОИСК И СОРТИРОВКА МАССИВОВ

Цель лабораторной работы: изучить способы сортировки и поиска в массивах записей и файлах.

4.1. Организация работы с базами данных

Для создания и обработки всевозможных баз данных широко применяются массивы структур.

Обычно база данных накапливается и хранится на диске. К ней часто приходится обращаться, обновлять, перегруппировывать. Работа с базой может быть организована двумя способами:

1. Внесение изменений и поиск осуществляется прямо на диске, используя специфическую технику работы со структурами в файлах. При этом временные затраты на обработку данных (поиск, сортировку) значительно возрастают, но нет ограничений на оперативную память.

2. Считывание в начале работы всей базы (или необходимой ее части) в массивы структур. При этом обработка производится в оперативной памяти, что значительно увеличивает скорость, однако требует больших затрат памяти.

Наиболее частыми операциями при работе с базами данных являются «поиск» и «сортировка». При этом алгоритмы решения этих задач существенно зависят от того, организованы структуры в массивы или размещены на диске.

Обычно структура содержит некое ключевое поле (ключ), по которому ее можно найти. Ключом может, например, служить фамилия или номер телефона или адрес. Основное требование к ключу в задачах поиска состоит в том, чтобы операция проверки на равенство была корректной. Поэтому, например, в качестве ключа не следует выбирать действительное число, т.к. из-за всегда возможной ошибки округления поиск нужного ключа может оказаться безрезультатным, хотя этот ключ в массиве имеется.

4.2. Поиск в массиве структур

Задача поиска требуемого элемента в массиве структур *a[N]* заключается в нахождении индекса *i*, удовлетворяющего условию a[i].key=x. Здесь ключ *key* выделен в отдельное поле, x - аргумент поиска того же типа что и *key*. После нахождения *i* обеспечивается доступ ко всем другим полям найденной структуры a[i].

Линейный поиск используется, когда нет никакой дополнительной информации о разыскиваемых данных. Он представляет собой последовательный перебор массива до обнаружения требуемого ключа или до конца, если ключ не обнаружен:

```
int i;
 for (i=0; i<N && A[i].key != k; i++);
 if (A[i].key != k) return < элемент не найден >;
```

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

```
        else return i;
```

Видно, что на каждом шаге требуется увеличивать индекс и вычислять логическое выражение. А можно ли уменьшить затраты на поиск? Единственная возможность – попытаться упростить логическое выражение с помощью введения вспомогательного элемента - *барьера*, который предохраняет от перехода за пределы массива:

```
int i=0;  A[N+1].key=k;
 while (A[i].key != k) i++;
  if (i==N+1) return < элемент не найден >;
        else return i;
```

Поиск делением пополам используется, если данные упорядочены, например, по возрастанию ключа *key*. Поиск осуществляется следующим образом. Берется средний элемент *m*. Если a[m]. key<x, то все элементы $i \le m$ исключаются из дальнейшего поиска, иначе - исключаются все $i > m$:

```
int i=0, j=N-1, m;
while(i<j)
        {
    m=(i+j)/2;
    if (A[m].key<k) i=m+1; else j=m;
        }
if (A[i].key!=k) return < элемент не найден >;
        else return i;
```

В этом алгоритме отсутствует проверка внутри цикла совпадения a[m].k=x. На первый взгляд это кажется странным, однако тестирование показывает, что в среднем выигрыш от уменьшения количества проверок превосходит потери от нескольких «лишних» вычислений до выполнения условия $i=j$.

4.3. Сортировка массивов

Под сортировкой понимается процесс перегруппировки элементов массива, приводящий к их упорядоченному расположению относительно ключа.

Цель сортировки – облегчить последующий поиск элементов. Метод сортировки называется устойчивым, если в процессе перегруппировки относительное расположение элементов с равными ключами не изменяется. Основное условие при сортировке массивов – не вводить дополнительных массивов, т.е. все перестановки элементов должны выполняться в исходном массиве. Сортировку массивов принято называть **внутренней** в отличие от сортировки файлов, которую называют **внешней**.

Методы внутренней сортировки классифицируются по времени их работы. Хорошей мерой эффективности может быть число сравнений ключей - *C* и число пересылок элементов - *P*. Эти числа являются функциями *C(n), P(n)* от числа сортируемых элементов *n*. **Быстрые** (но сложные) алгоритмы сортировки требуют (при $n \to \infty$) порядка $n \log n$ сравнений, **прямые** (простые) методы - n^2.

Прямые методы коротки, просто программируются, быстрые, усложненные, методы требуют меньшего числа операций, но эти операции обычно сами более

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

сложны, чем операции прямых методов, поэтому для достаточно малых n $(n \le 50)$ прямые методы работают быстрее. Значительное преимущество быстрых методов (в $n/log(n)$ раз) начинает проявляться при $n \gtrsim 100$.

Среди *простых* методов наиболее популярны:

1. **Метод прямого обмена** (пузырьковая сортировка).

```
for (i=0; i<N-1; i++)
  for (j=i+1; j<N; j++)
    if (A[i].key > A[j].key) { r=A[i];  A[i]=A[j];  A[j]=r; }
```

2. **Метод прямого выбора**:

```
for (i=0;i<N-1;i++)
                        {
            m=i;
          for (j=i+1; j<N; j++)
            if (A[j].key < A[m].key) m=j;
        r=A[m];  A[m]=A[i];  A[i]=r;
                        }
```

Реже используются:

3. Сортировка с помощью прямого (двоичного) включения;
4. **Шейкерная** сортировка (модификация пузырьковой).

Улучшенные методы сортировки:

1. **Метод Д. Шелла** (1959), усовершенствование метода прямого включения.
2. Сортировка с помощью дерева, метод **HeapSort**, Д. Уильямсон (1964).
3. Сортировка с помощью разделения, метод **QuickSort**, Ч. Хоар (1962), улучшенная версия пузырьковой сортировки. На сегодняшний день это самый эффективный метод сортировки.

Идея метода разделения **QuickSort** в следующем. Выбирается значение ключа среднего m-го элемента x=a[m].key. Массив просматривается слева направо до тех пор, пока не будет обнаружен элемент a[i].key>x. Затем массив просматривается справа налево, пока не будет обнаружен a[j].key<x. Элементы a[i] и a[j] меняются местами. Процесс просмотра и обмена продолжается до тех пор, пока i не станет больше j. В результате массив окажется разбитым на левую часть $a[l]$, $1 \le l \le j$ с ключами меньше (или равными) x и правую $a[p]$, $i \le p \le n$ с ключами больше (или равными) x.

Алгоритм такого разделения очень прост и эффективен:

```
 i=0; j=N-1; x=A[(L+R)/2].key;
  while (i<=j)
{
    while (A[i].key<x) i++;
 while (A[j].key>x) j--;
    if (i<=j)  {
      w=A[i];
      A[i]=A[j];
```

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

```
A[j]=w;
      i++; j--;
       }
}
```

Чтобы отсортировать массив, остается применять алгоритм разделения к левой и правой частям, затем к частям частей и так до тех пор, пока каждая из частей не будет состоять из одного единственного элемента. Алгоритм получается итерационным. На каждом этапе возникают две задачи по разделению. К решению одной из них можно приступить сразу, для другой следует заполнить начальные условия в список (номер разделения, границы и отложить ее решение до момента окончания сортировки выбранной половины. В нижеприведенном алгоритме для организации списка введен массив *Stak*.

Алгоритм этого метода можно записать следующим образом:

```
 struct TStack
 { int Left, Right; } Stack[20];
TMas w;
int i, j, x, s=0, L, R;
   Stack[0].Left=0;  Stack[0].Right=N-1;
while (s != -1)
 {
  L=Stack[s].Left;  R=Stack[s].Right; s--;
 while (L < R)
  {
    i=L;  j=R;  x=A[(L+R)/2].key;
  while (I <= j)
   {
     while (A[i].key < x) i++;
     while (A[j].key > x) j--;
   if (i<=j) {
     w=A[i];  A[i]=A[j];  A[j]=w;
       i++;  j--;
            }
    }
 if ((j-L)<(R-i))  // Выбор более короткой части
       {
 if (i<R) {s++; Stack[s].Left=i; Stack[s].Right=R; }
        R=j;
        }
  else {
 if (L<j) {s++; Stack[s].Left=L; Stack[s].Right=j; }
     L=i;
         }
 }
}
```

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

Сравнение методов сортировок показывает, что при n>100 наихудшим является метод пузырька, метод QuickSort в 2-3 раза лучше, чем HeapSort, и в 3-7 раз лучше, чем метод Шелла.

4.4. Индивидуальные задания

Написать программу, записывающую в файл и читающую из файла массив из структур. Написать следующие функции и организовать их вызов: функцию линейного поиска в файле; функцию сортировки массива и файла методами прямого выбора и QuickSort; функцию двоичного поиска в отсортированном массиве.

1. В магазине формируется список лиц, записавшихся на покупку товара. Каждая запись в этом списке содержит: порядковый номер, Ф.И.О., домашний адрес покупателя и дату постановки на учет. Удалить из списка все повторные записи, проверяя Ф.И.О. и домашний адрес. Ключ: дата постановки на учет.

2. Список товаров, имеющихся на складе, включает в себя наименование товара, количество единиц товара, цену единицы и дату поступления товара на склад. Вывести в алфавитном порядке список товаров, хранящихся больше месяца, стоимость которых превышает 100 000 р. Ключ: наименование товара.

3. Для получения места в общежитии формируется список студентов, который включает Ф.И.О. студента, группу, средний балл, доход на члена семьи. Общежитие в первую очередь предоставляется тем, у кого доход на члена семьи меньше двух минимальных зарплат, затем остальным в порядке уменьшения среднего балла. Вывести список очередности предоставления мест в общежитии. Ключ: доход на члена семьи.

4. В справочной автовокзала хранится расписание движения автобусов. Для каждого рейса указаны его номер, пункт назначения, время отправления и прибытия. Вывести информацию о рейсах, которыми можно воспользоваться для прибытия в пункт назначения раньше заданного времени. Ключ: время прибытия.

5. На междугородной АТС информация о разговорах содержит дату разговора, код и название города, время разговора, тариф, номер телефона в этом городе и номер телефона абонента. Вывести по каждому городу общее время разговоров с ним и сумму. Ключ: общее время разговоров.

6. Информация о сотрудниках фирмы включает: Ф.И.О., табельный номер, количество проработанных часов за месяц, почасовой тариф. Рабочее время свыше 144 ч считается сверхурочным и оплачивается в двойном размере. Вывести размер заработной платы каждого сотрудника фирмы за вычетом подоходного налога, который составляет 12% от суммы заработка. Ключ: размер заработной платы.

7. Информация об участниках спортивных соревнований содержит: наименование страны, название команды, Ф.И.О. игрока, игровой номер, возраст, рост, вес. Вывести информацию о самой молодой команде. Ключ: возраст.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

8. Для книг, хранящихся в библиотеке, задаются: регистрационный номер книги, автор, название, год издания, издательство, количество страниц. Вывести список книг с фамилиями авторов в алфавитном порядке, изданных после заданного года. Ключ: автор.

9. Различные цеха завода выпускают продукцию нескольких наименований. Сведения о выпущенной продукции включают: наименование, количество, номер цеха. Для заданного цеха необходимо вывести количество выпущенных изделий по каждому наименованию в порядке убывания количества. Ключ: количество выпущенных изделий.

10. Информация о сотрудниках предприятия содержит: Ф.И.О., номер отдела, должность, дату начала работы. Вывести списки сотрудников по отделам в порядке убывания стажа. Ключ: дата начала работы.

11. Ведомость абитуриентов, сдавших вступительные экзамены в университет, содержит: Ф.И.О., адрес, оценки. Определить количество абитуриентов, проживающих в г.Минске и сдавших экзамены со средним баллом не ниже 4.5, вывести их фамилии в алфавитном порядке. Ключ: Ф.И.О.

12. В справочной аэропорта хранится расписание вылета самолетов на следующие сутки. Для каждого рейса указаны: номер рейса, тип самолета, пункт назначения, время вылета. Вывести все номера рейсов, типы самолетов и времена вылета для заданного пункта назначения в порядке возрастания времени вылета. Ключ: пункт назначения.

13. У администратора железнодорожных касс хранится информация о свободных местах в поездах на ближайшую неделю в следующем виде: дата выезда, пункт назначения, время отправления, число свободных мест. Оргкомитет конференции обращается к администратору с просьбой зарезервировать m мест до города N на k-й день недели с временем отправления поезда не позднее t часов вечера. Вывести время отправления или сообщение о невозможности выполнить заказ в полном объеме. Ключ: число свободных мест.

14. Ведомость абитуриентов, сдавших вступительные экзамены в университет, содержит: Ф.И.О. абитуриента, оценки. Определить средний балл по университету и вывести список абитуриентов, средний балл которых выше среднего балла по университету. Первыми в списке должны идти студенты, сдавшие все экзамены на 5. Ключ: средний балл.

15. В радиоателье хранятся квитанции о сданной в ремонт радиоаппаратуре. Каждая квитанция содержит следующую информацию: наименование группы изделий (телевизор, радиоприемник и т. п.),марка изделия, дата приемки в ремонт, состояние готовности заказа (выполнен, не выполнен). Вывести информацию о состоянии заказов на текущие сутки по группам изделий**.** Ключ: дата приемки в ремонт.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

ТЕМА 5. ОРГАНИЗАЦИЯ ОДНОНАПРАВЛЕННОГО СПИСКА НА ОСНОВЕ РЕКУРСИВНЫХ ДАННЫХ В ВИДЕ СТЕКА

Цель лабораторной работы: изучить возможности работы со стеком в динамической памяти.

5.1. Определение стека

Стек – это структура данных, организованная по принципу «первый вошел – последний вышел». Образно это можно представить, как запаянную с одной стороны трубку, в которую закатываются шарики. Первый шарик всегда будет доставаться из трубки последним. Элементы в стек можно добавлять или извлекать только через его вершину. Программно стек реализуется в виде однонаправленного списка с одной точкой входа (вершиной стека).

Для работы со стеком вводится следующая структура:

```
struct TSel {
        TInf Inf;      // Информационная часть элемента стека
                       // например   typedef  int  TInf;
        TSel *A;    // Указатель на предыдущий элемент стека
          } ;
```

Для работы со стеком целесообразно организовывать библиотеку, в которую входят следующие подпрограммы.

Добавление элемента в стек

```
TSel *AddStask(TSel *Sp, TInf S)   // Sp – указатель на вершину стека
{                                  // S – добавляемая информация
      TSel *Spt;    // Временный указатель на элемент стека
     Spt = (TSel *) malloc(sizeof(TSel));   // Выделение памяти
                                            // под новый элемент стека

     Spt->Inf=S;  // Запись информационной части
     Spt->A=Sp; // Запись указателя на предыдущий элемент стека
   return Spt;      //  Новый указатель на вершину стека
}
```

Извлечение информации из стека с освобождением памяти

```
TSel *ReadStack(TSel *Sp, TInf *S)    // Sp - указатель на вершину стека
 {                                    // S - извлекаемая информация

      TSel *Spt;
```

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

```
      Spt=Sp;
      *S=Sp->Inf;
      Sp=Sp->A;
       free(Spt);    // Освобождение памяти
       return Sp;
 }
```

Чтение элемента стека с заданным номером *N* без освобождения памяти

```
bool ReadNStack(TSel *Sp, int N, TInf *S)  // Sp – указатель на вершину
      // стека, N - номер элемента стека, информацию из которого нужно
      // прочитать, S- прочитанная информация
{
 int i=1;
   while ((i!=N) && (Sp!=NULL))
    {
    i++;
    Sp=Sp->A;
    }
 if(Sp!=NULL)       // Если элемент найден, то
             {
     *S=Sp->Inf;   // сохранение информации
      return true;      // Элемент найден
              }
   else return false; // Элемент не найден
}
```

5.2. Сортировка однонаправленных списков

Часто, для ускорения поиска информации в списке, при выводе данных список упорядочивают (сортируют) по ключу.

Метод пузырьковой сортировки основан на проверке, и при необходимости, перестановке местами двух соседних элементов. Существует два способа перестановки элементов: обмен адресами и обмен информационными частями.

Первый способ основан на перестановке адресов двух соседних элементов, следующих за элементом с известным указателем. Первый элемент стека в этом случае не сортируется.

Функция перестановки адресов имеет вид

```
void RevAfter(TSel *Spi)
{
 TSel *Spt;
   Spt=Spi->A->A;
   Spi->A->A=Spt->A;
   Spt->A=Spi->A;
   Spi->A=Spt;
}
```

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

Функция сортировки для этого случая имеет вид

```
void SortBufAfter(TSel *Sp)
{
  TSel *Spt, *Spm;
 if (Sp->A->A==NULL) return;

  Spt=NULL;
    do {
     for (Spm=Sp; Spm->A->A != Spt; Spm=Spm->A)
          if (Spm->A->Inf > Spm->A->A->Inf)  RevAfter(Spm);
                Spt=Spm->A;
        } while (Sp->A->A != Spt);
}
```

Второй способ основан на обмене информации между ячейкой с текущим указателем и следующей за ней.

Функция обмена информацией имеет вид

```
void RevInf(TSel *Spi)
{
 TInf  Inf=Spi->Inf;
       Spi->Inf=Spi->A->Inf;
       Spi->A->Inf=Inf;
}
```

Функция сортировки для этого случая имеет вид

```
void SortBufInf(TSel *Sp)
{
  TSel *Spt, *Spm;
  Spt=NULL;
    do {
     for (Spm=Sp; Spm->A != Spt; Spm=Spm->A)
          if (Spm->Inf > Spm->A->Inf)  RevInf(Spm);
        Spt=Spm;
       } while (Sp->A != Spt);
}
```

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

5.3.Индивидуальные задания

Написать программу, создающую стек из случайных целых чисел и вычислить среднее арифметическое и среднеквадратичный разброс возле среднего. Реализовать сортировку стека двумя методами пузырька. Результат формирования и преобразования стека показывать в компонентах TlistBox. Выполнить индивидуальное задание в соответствии с полученным вариантом.

1. Создать стек из случайных целых чисел, лежащих в диапазоне –50 до +50 и преобразовать его в два стека. Первый должен содержать только положительные числа, а второй – только отрицательные. Порядок следования чисел должен быть сохранен.

2. Создать стек из случайных целых чисел и удалить из него записи с четными числами.

3. Создать стек из случайных целых чисел, лежащих в диапазоне от –10 до 10, и удалить из него записи с отрицательными числами.

4. Создать стек из случайных целых чисел и поменять местами крайние элементы.

5. Создать стек из случайных целых чисел и удалить элементы стека, заканчивающиеся на цифру 5.

6. Создать стек из случайных целых чисел и поменять местами элементы, содержащие максимальное и минимальное значения.

7. Создать стек из случайных целых чисел. Перенести в другой стек все элементы, находящиеся между вершиной и элементом с максимальным значением.

8. Создать стек из случайных целых чисел. Перенести в другой стек все элементы, находящиеся между вершиной и элементом с минимальным значением.

9. Создать стек из случайных чисел и определить, сколько элементов стека находится между минимальным и максимальным элементами и удалить их.

10. Создать стек из случайных чисел и определить, сколько элементов стека имеют значения меньше среднего значения от всех элементов стека и удалить эти элементы.

11. Создать стек из случайных чисел и поменять местами минимальный и максимальный элементы.

12. Создать стек из случайных целых чисел и из него сделать еще два стека. В первый поместить все четные, а во второй - нечетные числа.

13. Создать стек из случайных целых чисел в диапазоне от 1 до 10, определить наиболее часто встречающееся число и удалить все элементы стека, содержащие это число.

14. Создать стек из случайных целых чисел и удалить из него каждый второй элемент.

15. Создать стек из 10 произвольных строк. Замкнуть его в кольцо и, задав произвольное целое число (n), организовать выбывание элементов кольца через n шагов по кольцу. Определить последнюю оставшуюся строку. Аналог игры «На золотом крыльце сидели .....».

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

16. Создать стек из произвольного числа строк и реверсировать его, т.е. порядок следования элементов стека должен быть обратным.

17. Создать стек из произвольного числа строк и упорядочить его элементы в алфавитном порядке, используя метод “пузырька” (можно менять местами только два соседних элемента).

18. Создать два стека из упорядоченных по возрастанию целых случайных чисел. Объединить их в один стек с сохранением упорядоченности по возрастанию значений.

19. Создать стек из произвольного числа строк. Из него создать новый стек, куда бы входили в алфавитном порядке буквы, входящие в строки первого стека.

20. Создать стек из произвольного числа строк. Из него создать новый стек, в котором бы строки располагались в порядке увеличения числа символов в строке.

21. Создать стек из произвольного числа строк. Из него создать новый стек, в котором бы строки располагались в алфавитном порядке.

22. Создать стек из произвольного числа случайных комплексных чисел. Определить, сколько элементов стека имеют модуль комплексного числа больше среднего значения.

23. С помощью стека создать игру – «Минная дорожка». В стек случайным образом занести 100 нулей и 10 единиц. 1 – означает наличие мины, 0 – ее отсутствие на дорожке. Игрок должен последовательно вводить произвольные числа от 1 до 10, пока не достигнет дна стека. Если на его пути встречаются мины (значение 1), он проигрывает и игра начинается сначала.

24. Создать два упорядоченных по алфавиту стека со словами строкового типа. Объединить их в один стек с сохранением упорядоченности по алфавиту.

25. Создать три стека со случайными целыми числами без перекрытия их значений. Объединить их в один стек, соединив дно одного с вершиной другого и т.д.

26. Создать стек, содержащий координаты точек на плоскости. Найти наиболее удаленные между собой точки на плоскости.

27. Создать стек, содержащий координаты точек на плоскости. Найти геометрический центр этих точек и найти точку, к нему ближайшую.

28. Создать стек, содержащий координаты точек на плоскости. Найти геометрический центр этих точек и исключить из стека элемент, наиболее удаленный от центра.

29. Создать стек из случайных чисел. Преобразовать стек в кольцо. Посмотреть, к какому результату приведет циклическая операция, когда значение каждого элемента стека будет определяться как умноженное на 2, и с вычетом значения предыдущего элемента стека.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

ТЕМА 6. ОРГАНИЗАЦИЯ ОДНОНАПРАВЛЕННОГО И ДВУНАПРАВЛЕННОГО СПИСКОВ В ВИДЕ ОЧЕРЕДИ НА ОСНОВЕ РЕКУРСИВНЫХ ДАННЫХ

Цель лабораторной работы: изучить возможности работы со списками, организованными в виде очереди.

6.1. Очередь на основе односвязанного списка

Очередь по своей структуре очень похожа на стек, но только она работает по принципу «первый вошел – первый вышел». Ее образно можно представить в виде трубки, открытой с двух сторон. С одной стороны мы закатываем шарики, а с другой − извлекаем.

Для элемента очереди вводится такой же рекурсивный тип, как и для стека.

```
struct TSel {
 TInf Inf;      // Информационная часть элемента стека
                // например   typedef  int  TInf;
 TSel *A;    // Указатель на предыдущий элемент стека
   } ;
```

Добавление новой записи в конец очереди осуществляется следующей процедурой:

```
TSel *AddSpkK(TSel **SpN, TSel *SpK, TInf S)
{
  TSel *Spt;
      Spt = new TSel;
      Spt->Inf=S;
 if (*SpN==NULL) SpK = *SpN = Spt;
        else {
      SpK->A = Spt;
      SpK = Spt;
           }
   SpK->A =NULL;
 return SpK;
}
```

В результате первого обращения к этой подпрограмме (SpK==NULL) в списке размещается одна заявка. При этом инициируются два указателя: SpN - адрес 1-й записи, SpK - адрес последней.

Добавление новой записи в начало очереди:

```
TSel *AddSpkN(TSel *SpN, TSel **SpK, TInf S)
 {
```

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

```
  TSel *Spt;
     Spt = new TSel;
     Spt->Inf=S;
  if (*SpK==NULL)
                {
       Spt->A=NULL;
      *SpK = SpN = Spt;
                }
          else {
       Spt->A = SpN;
       SpN = Spt;
                }
 return SpN;
 }
```

Прочесть и стереть информацию из начала очереди можно с помощью следующей процедуры:

```
TSel *ReadSpkN(TSel *SpN,TSel **SpK, TInf *S)
{
       TSel *Spt;
       *S=SpN->Inf;
       Spt=SpN;
       SpN=SpN->A;
        delete(Spt);
       if (SpN==NULL) *SpK=NULL;
return SpN;
}
```

Распечатать очередь можно следующим образом:

```
        …
TInf S;
 while (SpN != NULL) {
      SpN=ReadSpkN(SpN,&SpK,&S) ;
   Memo1->Lines->Add(IntToStr(S));
                        }
```

Чтение и удаление элемента из середины однонаправленного списка сделать просто, если известен адрес ячейки Spi, находящейся **перед** удаляемой:

```
bool  ReadSpkI(TSel *Spi, TInf *S)
{
  if (Spi->A == NULL) return false;
           else {
          TSel *Spt;
           Spt=Spi->A;
            *S=Spt->Inf;
```

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

```
            Spi->A=Spt->A;
             delete(Spt);
             return true;
                }
}
```

Прочитать элемент можно следующим образом:

```
        …
TInf S;
  bool bl=ReadSpkI(SpN, &S);
    if (bl) Memo1->Lines->Add(IntToStr(S));
```

Добавление элемента в середину однонаправленного списка после элемента с адресом *Spi* :

```
TSel *AddSpkI(TSel *SpK, TSel *Spi, TInf S)
{
  TSel *Spt;
     Spt = new TSel;
     Spt->Inf=S;
 if (Spi->A==NULL)
              {
     Spi->A=Spt;
     Spt->A=NULL;
     SpK=Spt;
              }
         else {
      Spt->A = Spi->A;
      Spi->A = Spt;
              }
 return SpK;
}
```

Добавить элемент можно следующим образом:

```
        …
SpK=AddSpkI(SpK, SpN, S);
```

6.2. Двухсвязанные списки

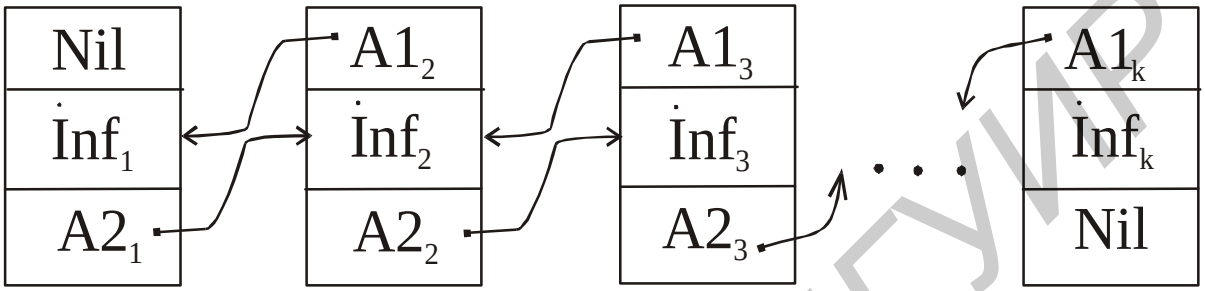
Двухсвязанные списки организуются, когда требуется просматривать список как в одном, так и в обратном направлении. Мы видели, что в однонаправленном списке довольно просто удалить (вставить) новую ячейку после заданной, но довольно сложно удалить саму заданную ячейку или предыдущую. Эта проблема легко решается, если ввести рекурсивную структуру с двумя адресными ячейками:

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

```
struct TSelD {
        TInf Inf;
        TSelD *A1;
        TSelD *A2;
          } ;
```

В A1 засылается адрес предыдущего элемента, в A2 - последующего, причем в первой ячейке с адресом spdm1 - значение A1=Nil, в последней ячейке с адресом spdmk - значение A2=Nil:



Создание двухсвязанного пустого списка с двумя метками входа реализуется с помощью следующей функции:

```
void AddNew(TSelD **SdN, TSelD **SdK)
{
  *SdN = new TSelD; *SdK = new TSelD;
  (*SdN)->A1=NULL;  (*SdN)->A2=*SdK;
  (*SdK)->A1=*SdN;   (*SdK)->A2=NULL;
}
```

Добавление элемента после заданного в двухсвязанный список:

```
void AddSpdAfter(TSelD *Sdi, TInf S)
{
   TSelD *Sdt;
      Sdt = new TSelD;
      Sdt->Inf=S;

       Sdi->A2->A1 = Sdt;
      Sdt->A2 = Sdi->A2;
      Sdt->A1 = Sdi;
      Sdi->A2 = Sdt;
}
```

Добавление элемента перед заданным:

```
void AddSpdBefor(TSelD *Sdi, TInf S)
{
   TSelD *Sdt;
      Sdt = new TSelD;
      Sdt->Inf=S;
       Sdi->A1->A2=Sdt;
```

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

```
        Sdt->A1 = Sdi->A1;
        Sdi->A1 = Sdt;
        Sdt->A2 = Sdi;
}
```

Чтение и удаление элемента с адресом spdi:

```
bool  ReadSpdI(TSelD *Sdi, TInf *S)
{
       if ((Sdi->A1 == NULL) || (Sdi->A2 == NULL)) return false;
           else {
            *S=Sdi->Inf;
             Sdi->A1->A2=Sdi->A2;
             Sdi->A2->A1=Sdi->A1;
             delete(Sdi);
           return true;
              }
}
```

Удаление всего списка:

```
void DelSpd(TSelD **SdN, TSelD **SdK)
{
  TSelD *Sdt;

    while (*SdN != NULL){
        Sdt= *SdN;
        *SdN=(*SdN)->A1;
        delete(Sdt);
                  }
     *SdK=NULL;
}
```

6.3. Сортировка очереди слиянием

Допустим, что есть два отсортированных в порядке возрастания списка *sq1, sqk, sr1, srk*. Построим алгоритм их слияния в один отсортированный список *spk*:

```
  void SlipSpk(TSel **SpN, TSel *SpQ, TSel *SpR)
   {
     TSel *Spt;
   if (SpQ->Inf < SpR->Inf) {
             *SpN=SpQ;
             SpQ=SpQ->A;
                      }
                 else {
             *SpN=SpR;
             SpR=SpR->A;
                      }
            Spt=*SpN;
```

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

```
 while ((SpQ != NULL) && (SpR != NULL))
 {
    if (SpQ->Inf < SpR->Inf) {
            Spt->A=SpQ;
            SpQ=SpQ->A;
                    }
                 else {
            Spt->A=SpR;
            SpR=SpR->A;
                    }
     Spt=Spt->A;
 }

 while (SpQ != NULL){
       Spt->A=SpQ;
       SpQ=SpQ->A;
       Spt=Spt->A;
                          }

 while (SpR != NULL){
       Spt->A=SpR;
       SpR=SpR->A;
       Spt=Spt->A;
                           }
  Spt->A=NULL;
 }
```

Разбиение списка на два списка:

```
void Div2Spk(TSel *SpN, TSel **SpQ, TSel **SpR)
  {
      TSel *SptQ, *SptR;
     *SpQ = *SpR = NULL;

      if  (SpN != NULL) {
      *SpQ =SpN;
       SpN = SpN->A;
                    }

      if  (SpN != NULL) {
        *SpR =SpN;
        SpN = SpN->A;
                    }
    SptQ = *SpQ;
    SptR = *SpR;
```

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

```
    while (SpN != NULL){
       SptQ->A  = SpN;
       SptQ = SptQ->A;
        SpN = SpN->A;
      if (SpN != NULL){
       SptR->A  = SpN;
       SptR = SptR->A;
        SpN = SpN->A;
                  }
                   }
    SptQ->A  = NULL;
    SptR->A  = NULL;
  }
```

Сортировка очереди слиянием (рекурсивная):

```
void SotrSlip(TSel **SpN)
 {
    TSel *SpQ, *SpR;
  if  ((*SpN)->A != NULL){
    Div2Spk(*SpN, &SpQ, &SpR);
     SotrSlip(&SpQ);
     SotrSlip(&SpR);
      SlipSpk(SpN,SpQ,SpR);
                  }
 }
```

6.4. Индивидуальные задания

Во всех задачах создать список из случайных чисел в виде очереди и отсортировать рекурсивным методом слияния, после чего выполнить задание в соответствии с вариантом. Отображать список в компонентах TListBox.

1. Создать очередь из случайных целых чисел. Найти минимальный элемент и сделать его первым.

2. Создать две очереди из случайных целых чисел. В первой найти максимальный элемент и за ним вставить элементы второй очереди.

3. Создать двухсвязанный список из случайных целых чисел. Удалить из списка все элементы, находящиеся между максимальным и минимальным.

4. Упорядочить элементы двухсвязанного списка случайных целых чисел в порядке возрастания методом “пузырька”, когда можно переставлять местами только два соседних элемента.

5. Представить текст программы в виде двухсвязанного списка. Задать номера начальной и конечной строк. Этот блок строк следует переместить в заданное место списка.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

7. Создать двухсвязанный список из случайных целых чисел. Из элементов, расположенных между максимальным и минимальным, создать первое кольцо. Остальные элементы должны составить второе кольцо.

8. Создать двухсвязанный список из случайных целых, положительных и отрицательных чисел. Из этого списка образовать два, первый из которых должен содержать отрицательные числа, а второй – положительные. Элементы списков не должны перемещаться в памяти.

9. Создать двухсвязанный список из строк программы. Преобразовать его в кольцо. Организовать видимую в компоненте Memo циклическую прокрутку текста программы.

10. Создать два двухсвязанных списка из случайных целых чисел. Вместо элементов первого списка, заключенных между максимальным и минимальным, вставить второй список.

11. Создать двухсвязанный список из случайных целых чисел. Удалить из списка элементы с повторяющимися более одного раза значениями.

12. Создать двухсвязанный список и поменять в нем элементы с максимальным и минимальным значениями, при этом элементы не должны перемещаться в памяти.

13. Создать двухсвязанный список из нарисованных вами картинок. Преобразовать его в кольцо и организовать его циклический просмотр в компоненте Image.

14. Создать двухсвязанный список из случайных чисел. Преобразовать его в кольцо. Предусмотреть возможность движения по кольцу в обе стороны с отображением местоположения текущего элемента.

15. Создать двухсвязанный список из текста вашей программы и отобразить его в TListBox. Выделить в TListBox часть строк и обеспечить запоминание этих строк. Далее выделить любую строку и нажать кнопку, которая должна обеспечивать перемещения выделенных ранее строк перед текущей строкой. При этом в TListBox должны отображаться строки из двухсвязанного списка.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

ТЕМА 7. ИСПОЛЬЗОВАНИЕ СТЕКА ДЛЯ ПРОГРАММИРОВАНИЯ АЛГОРИТМА ВЫЧИСЛЕНИЯ АЛГЕБРАИЧЕСКИХ ВЫРАЖЕНИЙ

Цель лабораторной работы: изучить возможности работы со списками и очередями.

7.1. Задача вычисления арифметических выражений

Одной из задач при разработке трансляторов является задача расшифровки арифметических выражений, например:

$$r:=(a+b) * (c+d)-e;$$

В выражении ***a+b*** ***a*** и ***b*** – операнды, **+** – операция. Такая запись называется **инфиксной** формой. Возможны также обозначения ***+ab*** – **префиксная**, ***ab+*** – **постфиксная** форма. В наиболее распространенной инфиксной форме для указания последовательности выполнения операций необходимо расставлять скобки. Польский математик Я. Лукашевич обратил внимание на тот факт, что при записи выражений в постфиксной форме скобки не нужны, а последовательность операндов и операций удобна для расшифровки, основанной на применении эффективных методов. Поэтому постфиксная запись выражений получила название **обратной польской записи**. Например, в ОПЗ вышеприведенное выражение выглядит следующим образом:

$$r=ab+cd+ * e-;$$

Алгоритм вычисления такого выражения основан на использовании стека. При просмотре выражения слева направо каждый операнд заносится в стек. В результате для каждой встреченной операции относящиеся к ней операнды будут двумя верхними элементами стека. Берем из стека эти операнды, выполняем очередную операцию над ними и результат помещаем в стек.

Алгоритм **преобразования выражения из инфиксной формы в форму обратной польской записи** (постфиксную) был предложен Дейкстрой. При его реализации вводится понятие стекового приоритета операций:

Операции	***)(***	***+ –***	**** /***	***^***	***(^ - возведение в степень)***
Приоритет	0	1	2	3	

Просматривается слева направо исходная строка символов, в которой записано выражение в **инфиксной форме**, причем операнды переписываются в выходную строку, в которой формируется постфиксная форма выражения, а знаки операций заносятся в стек следующим образом:

1. Если стек пуст, то операция записывается в стек.
2. Открывающаяся скобка дописывается в стек.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

3. Очередная операция выталкивает в выходную строку все операции из стека с большим или равным приоритетом.

4. Закрывающая скобка выталкивает все операции из стека до ближайшей открывающейся скобки в выходную строку, сами скобки уничтожаются.

5. Если после выборки из исходной строки последнего символа в стеке остались операции, то все они выталкиваются в выходную строку.

7.2. Порядок написания программы

Задание: Написать программу расшифровки и вычисления арифметических выражений с использованием стека.

Панель диалога будет иметь вид, показанный на рис. 7.1.

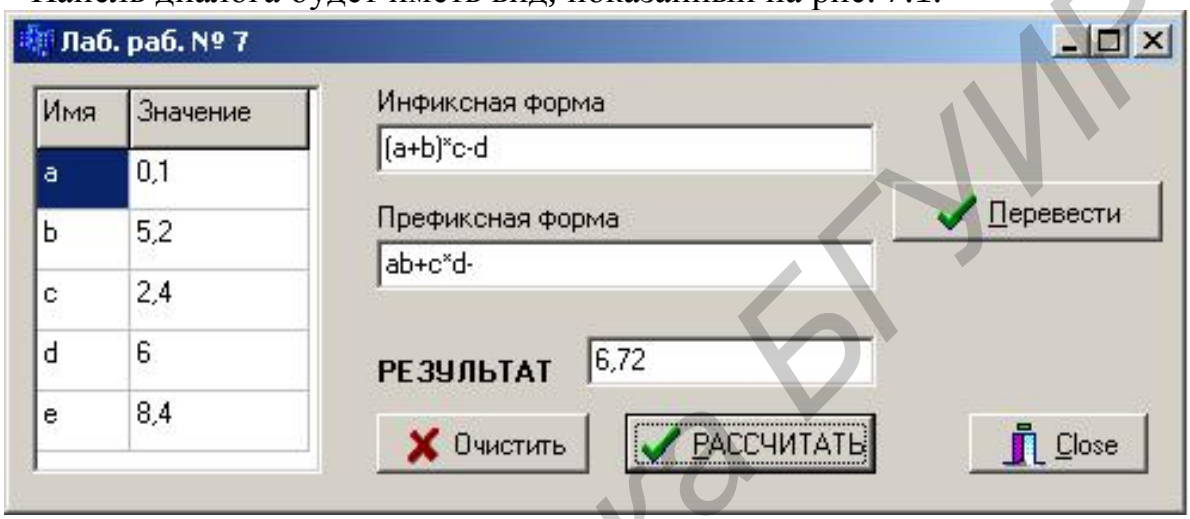


Рис. 7.1

Текст программы приведен ниже.

```
#include <vcl.h>
#pragma hdrstop
#include "lr07P.h"
#include <math.h>
//---------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------


  struct TSel {
          char Inf;
          TSel *A;
```

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

```
                } ;

  int SizeTSel = sizeof(TSel);
  AnsiString str;
  double mszn[201];   //[a..z]
   TSel  *stk;
    Set  <char, 0, 255> zn, zn2;

TSel *AddStask(TSel *Sp, char S) // Добавление элемента в стек
{
         TSel *Spt;
       Spt = (TSel *) malloc(SizeTSel);
        Spt->Inf=S;
        Spt->A=Sp;
        Sp=Spt;
         return Sp;
}

//---------------------------------------------------------------------

TSel *ReadStack(TSel *Sp, char *S) // Чтение элемента из стека
{
        TSel *Spt;
        Spt=Sp;
        *S=Sp->Inf;
        Sp=Sp->A;
         free(Spt);
         return Sp;
}

//---------------------------------------------------------------------
  double AV(AnsiString strp)   // Расчет арифметического выражения
{
   char ch,ch1,ch2;
   double op1,op2,rez;
   zn << '*' << '/' << '+' << '-' << '^';
   char chr='z'+1;

   for (int i=1; i<=strp.Length(); i++)
   {
     ch=strp[i];

   if (! zn.Contains(ch)) stk=AddStask(stk,ch);
                        else {
```

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

```
                    stk=ReadStack(stk,&ch1);
                    stk=ReadStack(stk,&ch2);
                      op1=mszn[int (ch1)];
                      op2=mszn[int (ch2)];
                   switch (ch){
              case '+' : rez=op2+op1;  break;
              case '-' : rez=op2-op1;  break;
              case '*' : rez=op2*op1;  break;
              case '/' : rez=op2/op1;   break;
              case '^' : rez=pow(op2,op1); break;
                       }
                   mszn[int (chr)]=rez;
                   stk=AddStask(stk,chr);
                   chr++;
                       }
  }
 return rez;
 }

//---------------------------------------------------------------------------

int  prior(char ch)   // Вычисление приоритета операций
{
   switch (ch){
    case '(' : case ')' : return 0;  break;
    case '+' : case '-' : return 1;  break;
    case '*' : case '/' : return 2;  break;
    case '^'            : return 3;  break;
            }
}

//---------------------------------------------------------------------------

AnsiString OBP(AnsiString stri)  // Перевод в постфиксную форму
{
 AnsiString strp="";
 TSel *stk=NULL;
 bool bl;
 int pc;
 char ch, ch1;
  zn2  << '*' << '/' << '+' << '-' << '^' << '(' <<')';
  int n=stri.Length();
     for (int i=1; i<=n; i++) {
            ch=stri[i];
```

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

```
        if (zn2.Contains(ch)){
          if (ch=='(') stk=AddStask(stk,ch);
                    else
           if (ch==')'){
            stk=ReadStack(stk,&ch);
           while (ch!='(') {
                    strp+=ch;
                    stk=ReadStack(stk,&ch);
                     }
                          }
                      else
                     if (stk==NULL) stk=AddStask(stk,ch); // Если стек пусть
                           else {   // Выталкивание менее приоритетных
                          pc=prior(ch);
                           do {
                              stk=ReadStack(stk,&ch1);
                          bl=prior(ch1)>=pc;
                                if (bl) strp+=ch1;
                             else stk=AddStask(stk,ch1);
                              } while ( (stk != NULL) && (bl));
                                stk=AddStask(stk,ch);
                                }   }
                                else  strp+=ch;
                          }
                          while (stk != NULL) {
                                        stk=ReadStack(stk,&ch);
                                       strp+=ch;
                                        }
  return strp;
}
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------

void __fastcall TForm1::FormCreate(TObject *Sender)
{
 Edit1->Clear();  Edit2->Clear(); Edit3->Clear();
  StringGrid1->Cells[0][0] ="Имя";   StringGrid1->Cells[1][0] ="Значение";
  StringGrid1->Cells[0][1] ="a";   StringGrid1->Cells[1][1] ="0,1";
  StringGrid1->Cells[0][2] ="b";   StringGrid1->Cells[1][2] ="5,2";
  StringGrid1->Cells[0][3] ="c";   StringGrid1->Cells[1][3] ="2,4";
  StringGrid1->Cells[0][4] ="d";   StringGrid1->Cells[1][4] ="6";
  StringGrid1->Cells[0][5] ="e";   StringGrid1->Cells[1][5] ="8,4";
}
```

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

```
//---------------------------------------------------------------------------

void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{
AnsiString stri=Edit1->Text;
 AnsiString strp=OBP(stri);
  Edit2->Text=strp;
}
//---------------------------------------------------------------------------


void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
Edit1->Clear();  Edit2->Clear();   Edit3->Clear();
}
//---------------------------------------------------------------------------

void __fastcall TForm1::BitBtn3Click(TObject *Sender)
{
char ch;
AnsiString strp=Edit2->Text;
  for (int i=1; i<=5; i++)
     {
   ch=StringGrid1->Cells[0][i][1];
   mszn[ int(ch)]=StrToFloat(StringGrid1->Cells[1][i]);
     }
  Edit3->Text=FloatToStr(AV(strp));
}
//---------------------------------------------------------------------------
```

7.3. Индивидуальные задания

В основе программы должны лежать два алгоритма: а) преобразование арифметического выражения из инфиксной формы в постфиксную (форму обратной польской записи); б) расшифровка и вычисление выражения в постфиксной форме.

У каждого варианта должно быть свое оригинальное арифметическое выражение, включающее операции + - ∗ / ^. Для контроля вычислить это же выражение обычным образом. Разработать удобный интерфейс ввода и вывода данных.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

ТЕМА 8. ПРОГРАММИРОВАНИЕ С ИСПОЛЬЗОВАНИЕМ ДЕРЕВЬЕВ НА ОСНОВЕ РЕКУРСИВНЫХ ТИПОВ ДАННЫХ

Цель лабораторной работы: изучить способы программирования алгоритмов обработки данных с использованием древовидных структур. Получить навыки работы с компонентом TTreeView

8.1. Понятие древовидной структуры

Древовидная модель оказывается довольно эффективной для представления динамических данных с целью быстрого поиска информации.

Древовидное размещение списка данных (*abcdefgkl*) можно изобразить, например, следующим образом:

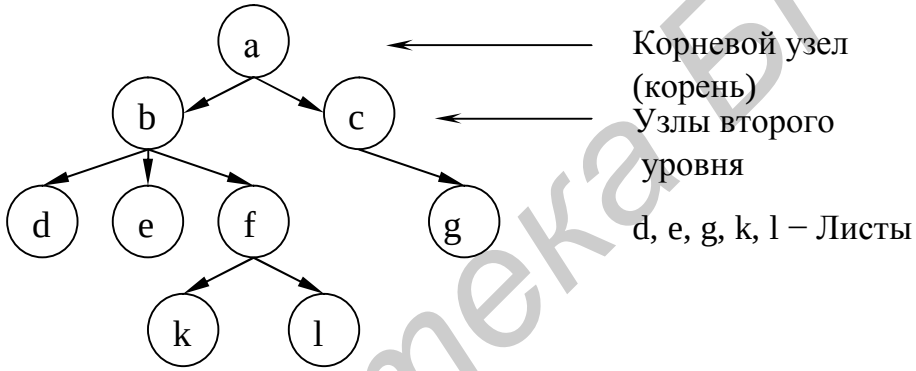


Рис. 8.1

Как видим, данные размещаются в узлах дерева, соединенных направленными дугами (ветвями). Если два узла соединены направленной дугой (*X*®*Y*), то узел *X* называется **предшественником** (родителем), а узел *Y* – **преемником** (дочерью). Деревья имеют единственный узел (корень), *у* которого нет родителя. Узел, не имеющий дочерей, называется **листом**. **Внутренний** узел – это узел, не являющийся листом или корнем. **Порядок узла** – количество его дочерних узлов. **Степень дерева** – максимальный порядок его узлов. **Глубина узла** равна числу его предков плюс 1. **Глубина дерева** – это наибольшая глубина его узлов. Дерево, представленное выше, имеет степень 3 (троичное), глубину 4, корневой узел **a**.

Для реализации древовидных структур данных степени m используется следующая конструкция рекурсивного типа данных:

```
struct TTree {
        TInf Inf;
       TTree *A1;        // A1 … Am – указатели на адреса,
           ...           // по которым расположены сестры
       TTree *Am;       // Если сестра отсутствует, то
           } ;          // соответствующий адрес равен NULL
```

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

Размещение данных в виде древовидной структуры, представленной на рис. 8.1, можно осуществить, например, следующим образом:

```
TTree *proot, *p;      // Указатели: на корень и текущий
                ...
proot = new TTree;  p = proot;    p->Inf= "a";  p->A2 = NULL;
p->A3 = new TTree;  p = p->A3;    p->Inf= "c";  p->A1 = NULL;
p->A2 = NULL; p->A3 = new TTree; p = p->A3;    p->Inf= "g";
p->A1 = NULL; p->A2 = NULL;
p->A3 = NULL; p = proot; p->A1 = new TTree;
p = p->A1;   p->Inf= "b"; p->A1 = new TTree; p->A1->Inf= "d";
p->A1->A1 = NULL; p->A1->A2 = NULL; p->A1->A3 = NULL;
p->A2 = new TTree;     p->A2->Inf= "e"; p->A2->A1 = NULL;
p->A2->A2 = NULL; p->A2->A3 = NULL;
p->A3 = new TTree; p = p->A3;
p->Inf= "f"; p->A2 = NULL; p->A1 = new TTree; p->A1->Inf= "k";
p->A1->A1=NULL; p->A1->A2=NULL; p->A1->A3=NULL;
p->A3 = new TTree;     p->A3->Inf= "l"; p->A3->A1 = NULL;
p->A3->A2 = NULL; p->A3->A3 = NULL;
```

После того как дерево заполнено информацией, его нужно уметь просмотреть, распечатать, осуществить поиск. Как видно из вышеприведенного примера, непосредственное заполнение даже небольшого дерева требует довольно громоздкой последовательности команд. Поэтому для работы с деревьями используют набор специфических алгоритмов. Последовательное обращение ко всем узлам называется **обходом дерева**. Следующая рекурсивная функция осуществляет такой обход с распечаткой каждого узла:

```
void WrtTree(TTree **p)    // p - указатель на текущий узел дерева
 {
  if (*p == NULL) return;    // Если дочь отсутствует, то выход
     < Print(p^.Inf); >          // При прямом обходе печать ставить здесь
  WrtTree((&(*p)->A1));
  WrtTree((&(*p)->A2));
    ...
 WrtTree((&(*p)->Am));
   < Print(p^.Inf); >        // При обратном обходе печать ставить здесь
 }
```

При вызове функции

```
WrtTree(&proot);
```

данные, изображенные на рисунке, будут распечатаны в следующем порядке:
Прямой обход (печать стоит вначале): *a, b, d, e, f, k, l, c, g.*
Обратный обход (печать стоит в конце): *d e k l f b g c a.*

Аналогично можно составить функцию нахождения информации в дереве по заданному ключу, который находится в поле Inf.key:

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

```
void PoiskTree(TTree *p, int k, bool *bl, TInf *Res)
 {
   if ((p != NULL) && (*bl != true)) {
       if (p->Inf.key != k) {
    PoiskTree(p->A1,k, bl, Res);
    PoiskTree(p->A2,k, bl, Res);
                …
    PoiskTree(p->Am,k, bl, Res);
                                }
                      else  {
                  *bl=true;
                  *Res =  p->Inf;
                                }
                                        }
 }
```

Вызов процедуры :

```
  bool b = false;
PoiskTree(proot, k, &b, &Res);
  if (b) < Print(Res); >    // Печать найденной информации
```

8.2. Компонент TTreeView

Компонент TTreeView предназначен для отображения ветвящихся иерархических структур в виде горизонтальных деревьев, например, каталогов файловой системы дисков. Основным свойством этого компонента является Items, которое представляет собой массив элементов типа TTreeNode, каждый из которых описывает один узел дерева. Всего в Items - count узлов. Рассмотрим некоторые методы класса TTreeNodes.

TTreeNode* __fastcall AddFirst(TTreeNode* Node, const AnsiString S) – создает первый дочерний узел в узле Node. Если Node=Nil, то создается корневой узел. S – это строка содержания узла. Функция возвращает указатель на созданный узел.

TTreeNode* __fastcall AddChild(TTreeNode* Node, const AnsiString S) – добавляет очередной дочерний узел к узлу Node.

TTreeNode* __fastcall AddChildFirst(TTreeNode* Node, const AnsiString S) – добавляет новый узел как первый из дочерних узлов узла Node.

void __fastcall Clear(void) – очищает список узлов, описанных в свойстве Items.

void __fastcall FullExpand(void) – раскрывает все узлы при отображении дерева.

8.3. Бинарное дерево поиска

Наиболее часто для работы со списками используется бинарное (имеющие степень 2) дерево поиска. Предположим, что у нас есть набор данных, упорядоченных по ключу:

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

key: 1, 6, 8, 10, 20, 21, 25, 30.

Если организовать древовидную структуру со следующим распределением ключей (*N=NULL*)

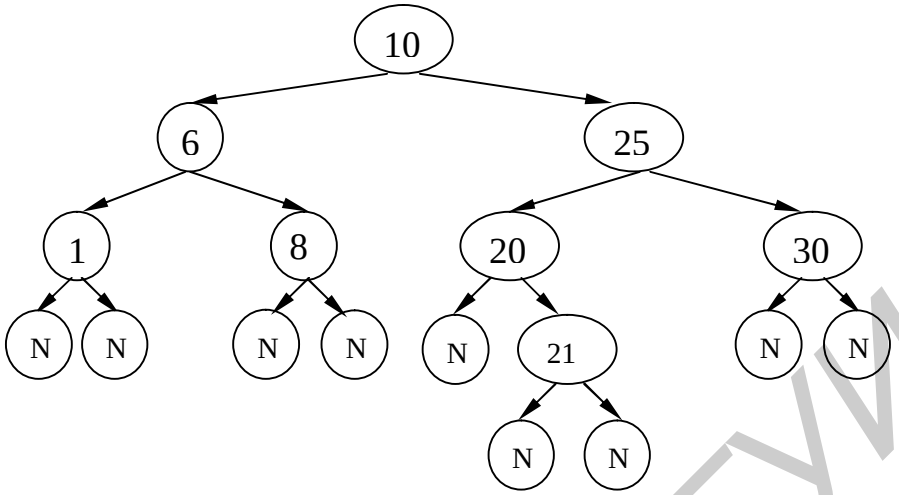


Рис. 8.2

то обход дерева в **симметричном порядке** организуется функцией

```
void WrtTree(TTree **p)    // p - указатель на текущий узел дерева
 {
  if (*p == NULL) return;     // Если дочь отсутствует, то выход
  WrtTree((&(*p)->A1));
  < Print(p^.Inf); > // При симметричном обходе печать ставить здесь
  WrtTree((&(*p)->A2));
}
```

При вызове этой функции будет напечатана информация в порядке возрастания ключа. Если в процедуре поменять местами A1 и A2, то информация будет напечатана в порядке убывания ключа.

В дереве на рис. 8.2 ключи расположены таким образом, что для любого узла значения ключа у левого преемника меньше, чем у правого. Таким образом, организованное дерево получило название **двоичное дерево поиска**. Ввиду его своеобразной организации эффективность поиска информации в такой динамической структуре данных сравнима с эффективностью двоичного поиска в массиве, т.е. $O(\log_2 n)$. Заметим, что двоичный поиск в линейном связанном списке организовать невозможно, а эффективность линейного поиска имеет порядок $O(n/2)$.

Конечно, оценка $O(\log_2 n)$ справедлива для **сбалансированного** дерева, т.е. такого, у которого узлы, имеющие только одну дочь, располагаются не выше двух последних уровней.

8.4. Основные операции с двоичным деревом поиска

При организации списков в виде двоичного дерева необходим пакет программ, реализующих следующие действия:

- поиск заданного ключа;
- поиск минимального (максимального) ключа;

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

- вставка нового значения ключа, не изменяя свойств дерева поиска;
- удаление заданного ключа;
- формирование дерева поиска;
- балансировка дерева.

Поиск в двоичном дереве по заданному ключу Inf.key:

```
void PoiskTree(TTree *p, int k, bool *bl, TInf *Res)
 {
   if ((p != NULL) && (*bl != true)) {
       if (p->Inf.key != k)   {
    PoiskTree(p->A1,k, bl, Res);
    PoiskTree(p->A2,k, bl, Res);
                              }
                    else      {
                     *bl=true;
                     *Res =  p->Inf;
                              }
                               }
 }
```

Поиск информации с минимальным (максимальным) ключом:

```
TInf MinKey(TTree *p)
 {
   while (p->A1 != NULL) p = p->A1;
   return p->Inf;
 }
```

При поиске максимального ключа нужно спуститься по правой ветке (p:=p^.A2) до самого правого листа.

Вставить новый элемент с ключом key, не совпадающим ни с одним из ключей:

```
void MakeList(TTree **p, TInf Inf)  // Создание нового листа дерева
{
 *p = new TTree;
 (*p)->Inf = Inf;
 (*p)->A1=NULL;
 (*p)->A2=NULL;
}
//---------------------------------------------------------------------------
void DobTree(TTree *proot, TInf Inf)  // Добавление листа в дерево
{
TTree  *p = proot, *q;
bool bl;

   while (p != NULL) {
```

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

```
      q = p;
      bl = ( Inf.key < p->Inf.key);
       if (bl) p = p->A1;
         else  p = p->A2;
                  }
   MakeList(&p, Inf);

    if (bl) q->A1=p; else  q->A2=p;
}
```

Построение дерева поиска:

Пусть имеется некоторый массив из n значений данных с разными ключами a:array[1..n] of Tinf. Построение дерева поиска можно осуществить следующим образом:

```
      MakeList(&proot,A[1]);
       for(int i=2; i<=n; i++) DobTree(proot,A[i]);
```

При случайном чередовании ключей в массиве **a** обычно формируется дерево, достаточно хорошо сбалансированное. Однако, если ключи в исходном массиве **a** частично упорядочены, то дерево поиска оказывается сильно разбалансированным и его эффективность для организации поиска оказывается сравнимой с линейным поиском в массиве.

Этот способ построения дерева поиска можно использовать для **печати элементов массива** (например расположенного в файле) **в порядке возрастания**. Для этого достаточно построить и распечатать полученное дерево.

Построение сбалансированного дерева поиска для заданного массива **a** можно осуществить, если этот массив предварительно отсортирован в порядке возрастания ключа.

Следующая рекурсивная процедура строит идеально сбалансированное дерево поиска по отсортированному массиву:

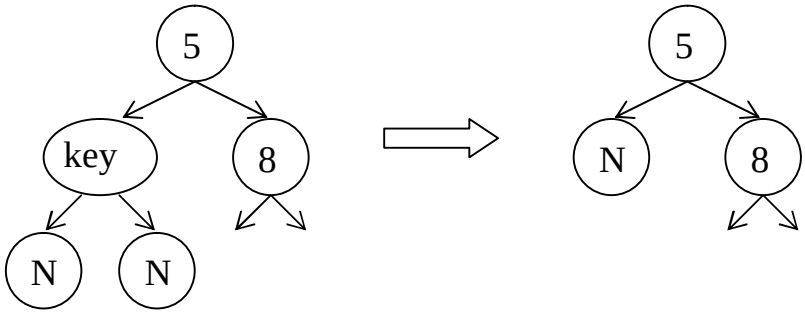
```
      void MakeBlns(TTree **p, int k, int l, TInf A[ ])
{
  if (k==l) { *p = NULL;
              return;
            }
           else {
            int m=(k+l)/2;
                *p = new TTree;
                 (*p)->Inf =A[m];
                 MakeBlns((&(*p)->A1),k,m,A);
                 MakeBlns((&(*p)->A2),m+1,l,A);
                }
 }
```

Библиотека БГУИР

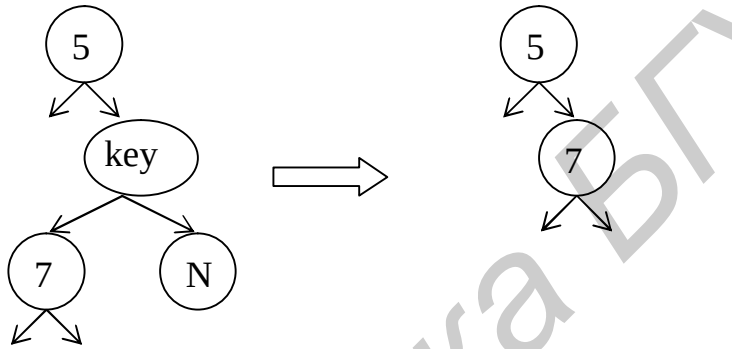
PDF created with pdfFactory Pro trial version www.pdffactory.com

Удаление узла с заданным ключом из дерева поиска, сохраняя его свойства. При решении этой задачи следует рассматривать три разные ситуации:

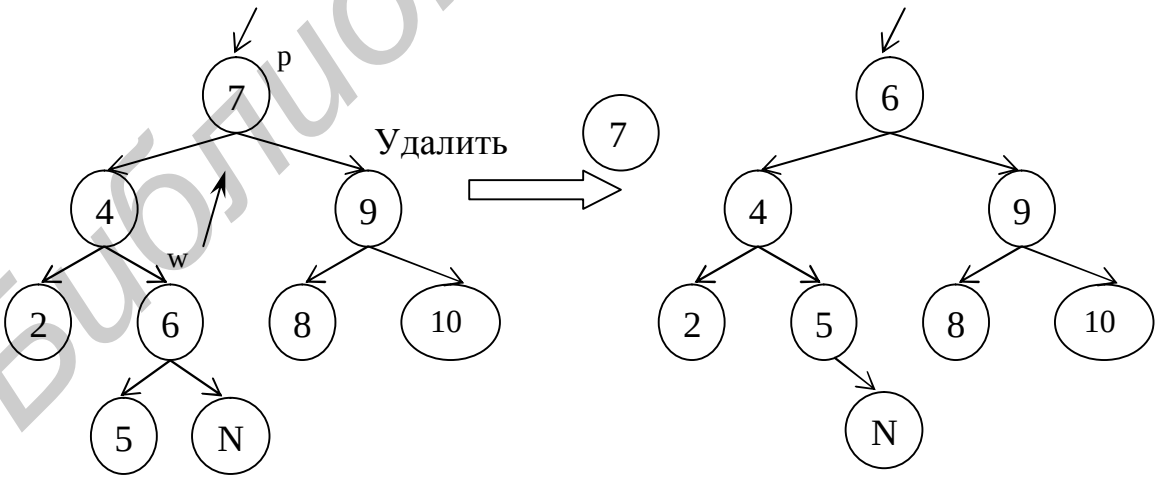
1. Удаление листа с ключом key:



2. Удаление узла, имеющего одну дочь:



3. Удаление узла, имеющего двух дочерей, значительно сложнее. Если *p* – исключаемый узел, то его следует заменить узлом *w*, который содержит наибольший ключ в левом поддереве p (либо наименьший ключ в правом поддереве). Такой узел *w* является либо листом, либо самым правым узлом поддерева p, у которого имеется только левая дочь:




```
void UdTree(TTree **proot, TInf Inf)
 {
 TTree  *p = *proot, *q = p, *w, *v;
  // Поиск удаляемого узла
  while ((p != NULL) && (p->Inf.key != Inf.key)) {
```

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

```
      q = p;
    if (p->Inf.key > Inf.key) p = p->A1;
                      else p = p->A2;
                                    }
    if (p == NULL) return; // Если узел не найден, то выход
              // Если узел не имеет дочерей
if ((p->A1 == NULL) && (p->A2 == NULL)){
            if (p == q) *proot = NULL;
             else
            if (q->A1 == p) q->A1 = NULL;
                     else q->A2 = NULL;
                                  }
else
              // Если узел имеет дочь слева
if (p->A1 == NULL) {
            if (p == q) *proot = p->A2;
             else
            if (q->A1 == p) q->A1 = p->A2;
                     else q->A2 = p->A2;
                }
else
              // Если узел имеет дочь справа
if (p->A2 == NULL) {
            if (p == q) *proot = p->A1;
             else
            if (q->A2 == p) q->A2 = p->A1;
                     else q->A1 = p->A1;
                }
else        // Если узел имеет двух дочерей
                  {
           w = p->A1;
             if (w->A2 == NULL) w->A2 = p->A2;
                else {
                  do {
                    v = w;
                    w = w->A2;
                     } while (w->A2 != NULL);


                     v->A2 = w->A1;
                     w->A1 = p->A1;
                     w->A2 = p->A2;
                        }
              if (p == *proot) *proot = w;
```

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

```
                else
            if (q->A1 == p) q->A1 = w;
                    else q->A2 = w;

                }
  delete(p); // Удаление узла
}
```

8.5. Порядок написания программы

Задание: В качестве примера рассмотрим проект, который создает дерево, отображает его в компонентах TTreeView и Memo и удаляет дерево. Панель диалога будет иметь вид, представленный на рис. 8.3.

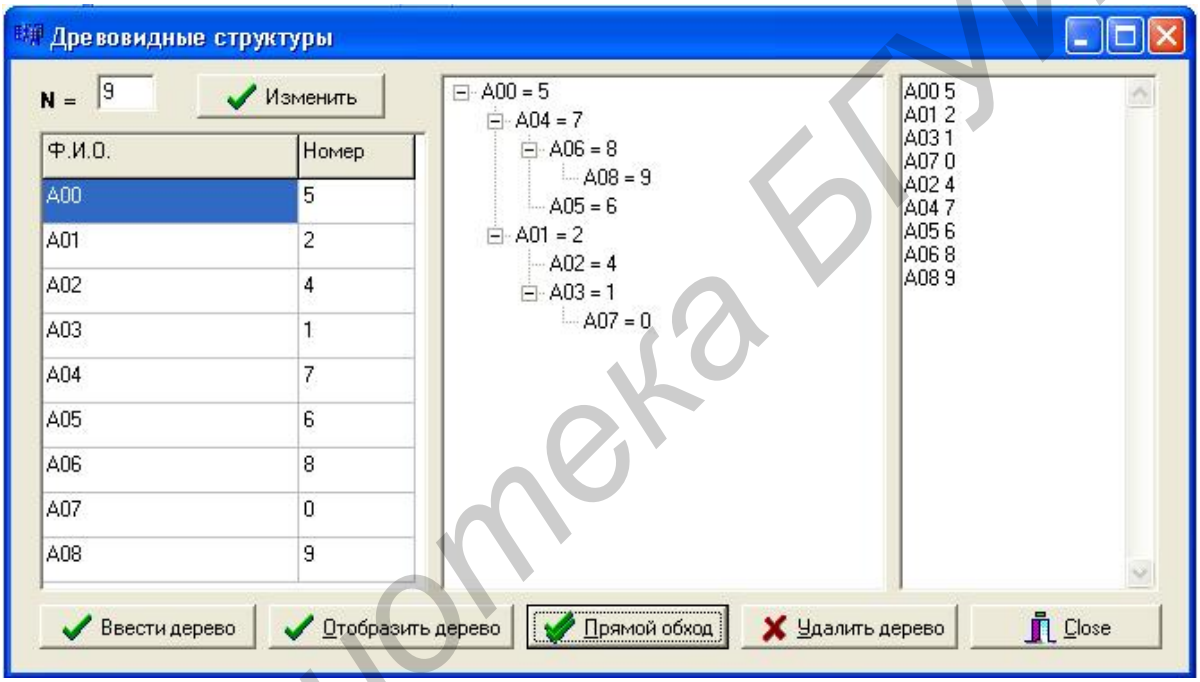


Рис. 8.3

Текст программы приведен ниже.

```
#include <vcl.h>
#pragma hdrstop

#include "LR8.h"
#include "lr8u.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;

int n = 9;

TTree *proot;
```

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

```
//---------------------------------------------------------------------------

__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}

//---------------------------------------------------------------------------

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  StringGrid1->Cells[0][0] = "Ф.В.О.";
  StringGrid1->Cells[1][0]="Ключ";
  StringGrid1->Cells[0][1]="A00";   StringGrid1->Cells[1][1]="5";
  StringGrid1->Cells[0][2]="A01";   StringGrid1->Cells[1][2]="2";
  StringGrid1->Cells[0][3]="A02";   StringGrid1->Cells[1][3]="4";
  StringGrid1->Cells[0][4]="A03";   StringGrid1->Cells[1][4]="1";
  StringGrid1->Cells[0][5]="A04";   StringGrid1->Cells[1][5]="7";
  StringGrid1->Cells[0][6]="A05";   StringGrid1->Cells[1][6]="6";
  StringGrid1->Cells[0][7]="A06";   StringGrid1->Cells[1][7]="8";
  StringGrid1->Cells[0][8]="A07";   StringGrid1->Cells[1][8]="0";
  StringGrid1->Cells[0][9]="A08";   StringGrid1->Cells[1][9]="9";
}
//---------------------------------------------------------------------------

void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{
  TInf A;
    A.fio = StringGrid1->Cells[0][1];
    A.key  = StrToInt(StringGrid1->Cells[1][1]);
     MakeList(&proot,A);
  for(int i=2; i<=n; i++){
    A.fio = StringGrid1->Cells[0][i];
    A.key  = StrToInt(StringGrid1->Cells[1][i]);
      DobTree(proot,A);
            }

}
//---------------------------------------------------------------------------

void __fastcall TForm1::BitBtn3Click(TObject *Sender)
{
int kl=-1;
```

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

```
  TreeView1->Items->Clear();
   ViewTree(proot,kl);
  TreeView1->FullExpand();
}
//---------------------------------------------------------------------------


void __fastcall TForm1::BitBtn4Click(TObject *Sender)
{
DeleteTree(&proot);
Memo1->Clear();
TreeView1->Items->Clear();
}
//---------------------------------------------------------------------------

void __fastcall TForm1::BitBtn5Click(TObject *Sender)
{
Memo1->Clear();
WrtTree(&proot);
}
//---------------------------------------------------------------------------

void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
n=StrToInt(Edit1->Text);
 StringGrid1->RowCount = n+1;
}
```

Текст заголовочного файла:

```
#ifndef lr8uH
#define lr8uH
#include <vcl.h>

//---------------------------------------------------------------------------

  struct TInf {
         int key;
        AnsiString fio;
             };



 struct TTree {
```

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

```
        TInf Inf;
        TTree *A1;
        TTree *A2;
          } ;

#endif

  void MakeList(TTree**, TInf);
  void DobTree(TTree*, TInf);
  void ViewTree(TTree*, int);
  void DeleteTree(TTree**);
  void WrtTree(TTree**);
```

Текст модуля:

```
#pragma hdrstop

#include "lr8u.h"
#include "lr8.h"
#include <alloc.h>

//---------------------------------------------------------------------------

#pragma package(smart_init)
//---------------------------------------------------------------------------

void MakeList(TTree **p, TInf Inf)
{
 *p = new TTree;
 (*p)->Inf = Inf;
 (*p)->A1=NULL;
 (*p)->A2=NULL;
}
//---------------------------------------------------------------------------

void DobTree(TTree *proot, TInf Inf)
{
TTree  *p = proot, *q;
bool bl;

    while (p != NULL) {
       q = p;
       bl = ( Inf.key < p->Inf.key);
        if (bl) p = p->A1;
```

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

```
        else  p = p->A2;
                    }
    MakeList(&p, Inf);

     if (bl) q->A1=p; else  q->A2=p;
}

//---------------------------------------------------------------------------

 void ViewTree(TTree *p, int kl)
 {
  if (p == NULL) return;
     if (kl == -1)
 Form1->TreeView1->Items->AddFirst(NULL,p->Inf.fio+" = "
   +IntToStr(p->Inf.key));
    else
Form1->TreeView1->Items->AddChildFirst(Form1->TreeView1->Items->
Item[kl],  p->Inf.fio+" = "+IntToStr(p->Inf.key));
    kl++;
   ViewTree(p->A1,kl);
   ViewTree(p->A2,kl);
    kl--;
 }
//---------------------------------------------------------------------------

 void DeleteTree(TTree **p
 {
  if (*p == NULL)return;
    DeleteTree(&(*p)->A1);
    DeleteTree(&(*p)->A2);
      delete(*p);
    *p=NULL;
 }
 //---------------------------------------------------------------------------

 void WrtTree(TTree **p) //
 {
  if (*p == NULL)return;
     Form1->Memo1->Lines->Add((*p)->Inf.fio+" "+(*p)->Inf.key);
   WrtTree((&(*p)->A1));
   WrtTree((&(*p)->A2));
 }
```

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

8.6. Индивидуальные задания

Создать проект для работы с деревом поиска, содержащий обработчики, которые должны:

- ввести информацию из компонента StringGrid в массив. Каждый элемент массива должен содержать строку текста и целочисленный ключ (например Ф.И.О. и номер паспорта);
- внести информацию из массива в дерево поиска;
- сбалансировать дерево поиска;
- добавить в дерево поиска новую запись;
- по заданному ключу найти информацию в дереве поиска и отобразить ее;
- удалить из дерева поиска информацию с заданным ключом;
- распечатать информацию прямым, обратным обходом и в порядке возрастания ключа;
- решить одну из следующих задач:

1. Поменять местами информацию, содержащую максимальный и минимальный ключи.
2. Подсчитать число листьев в дереве. (Лист – это узел, из которого нет ссылок на другие узлы дерева).
3. Удалить из дерева ветвь с вершиной, имеющей заданный ключ.
4. Определить максимальную глубину дерева, т.е. число узлов в самом длинном пути от корня дерева до листьев.
5. Определить число узлов на каждом уровне дерева.
6. Удалить из левой ветви дерева узел с максимальным значением ключа и все связанные с ним узлы.
7. Определить количество символов во всех строках дерева.
8. Определить число листьев на каждом уровне дерева.
9. Определить число узлов в дереве, в которых есть указатель только на один элемент дерева.
10. Определить число узлов в дереве, у которых есть две дочери.
11. Определить количество записей в дереве, начинающихся с определенной буквы (например “а”).
12. Найти среднее значение всех ключей дерева и найти строку, имеющую ближайший к этом значению ключ.
13. Найти запись с ключом, ближайшим к среднему значению между максимальным и минимальным значениями ключей.
14. Определить количество записей в левой ветви дерева.
15. Определить количество записей в правой ветви дерева.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

ТЕМА 9. ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

Цель лабораторной работы: ознакомиться с понятиями объекта и класса в Builder C++. Написать программу, использующую классы-предки и классы-потомки.

9.1. Понятие объекта и класса

Ключевым понятием ***объектно-ориентированного программирования (ООП)*** является ***объект***, который представляет собой особый структурированный тип переменных. Подобно обыкновенной структуре переменная типа **объект** под одним именем объединяет как данные различных типов (называемые, как и у записей, ***полями*** объекта), так функции обработки этих данных (называемые ***функциями-членами*** объекта).

В **C++** для описания объектов введен специальный тип - c***lass***. Синтаксис объявления класса:

```
class <имя класса>  : <классы - родители>
{
public:
 // Данные, доступные для внешнего использования
__publshed:
// Данные, которые видны в Инспекторе объектов
protected:
// Данные, доступные только для потомков данного класса
private:
// Данные, используемые только внутри данного класса
}
```

Работа с классами осуществляется, например, следующим образом:

```
class TMyCl
{
public:
int m;
int readme(void);
void writeme(int i);
};

TMyCl  ObjX;
ObjX.m=33;   int y= ObjX.readme();  ObjX.writeme(z);
```

Как видно, обращение к полям и методам объекта подобно обращению к полям записи. Однако функции-члены (методы) являются подпрограммами. В

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

описании объекта указывается лишь его заголовок, а его тело должно быть описано ниже:

```
int ObjX:: readme(void) { return m; }
void ObjX::writeme(int i) { m=i; }
```

В теле метода при обращении к полям и методам самого вызывающего их объекта имя объекта опускается, а при обращении к полям и методам других объектов – указывается явно. В каждом экземпляре объекта содержатся все его поля, но функции-члены класса хранятся в памяти в одной-единственной копии.

9.2. Создание и уничтожение объектов

Для инициализации значений полей объекта явно объявляют функцию-член, предназначенную для этого. Поскольку такая функция конструирует значение данного типа, она называется конструктором. Конструктор распознается по тому, что имеет то же имя, что и сам класс.

Например:

```
class date {
//……
date (int, int, int);
};
```

Конструктор вызывается в момент создания класса и предназначен для инициализации данных. Наличие конструктора в классе необязательно, но при его отсутствии необходимо использовать другие функции для задания начальных значений.

Для уничтожения динамически размещенных объектов класса и освобождения выделенной памяти используется деструктор. Имя деструктора представляет собой имя класса и стоящую перед ним тильду (~), например, для класса TMyCl деструктор имеет имя ~ TMyCl ().

9.3. Наследование и полиморфизм

Свойство ***наследования*** заключается в том, что любой класс может быть порожден от другого класса. Класс, от которого порождаются другие классы, называют “классом-родителем”, а класс, порожденный от другого класса, называется “классом-потомком”. Порожденный класс наследует поля, методы и свойства своего родителя и обогащает их новыми полями, методами и свойствами. Таким образом, принцип наследования позволяет эффективно использовать уже наработанный задел программ и на его основе создать классы для решения все более сложных задач.

Полиморфизм – это свойство классов, которое позволяет использовать одинаковое название метода для решения сходных, но несколько отличающихся в разных родственных классах задач. Обеспечивается это тем, что в классе-потомке одноименный метод переписывается по новому алгоритму, т.е. ***перекрывается***. В результате в объекте-родителе и объекте-потомке будут действовать два одноименных метода с разными алгоритмами.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

Метод потомка может вызывать методы предков. Для этого используют запись

<базовый класс> : : <метод базового класса>

В случае, если у потомка неизмененный родительский метод должен вызывать измененный метод потомка, вызываемый метод должен быть описан как ***виртуальный*** в родительском классе с помощью слова ***virtual***.

9.4. Пример написания программы

Рассмотрим задачу, включающую следующие подзадачи:

а) создать класс для отображения на форме прямоугольника, имеющего заданное положение и цвет. Класс должен имеет метод для перемещения прямоугольника по экрану;

б) на его основе описать класс-потомок для отображения изображения автомобиля, включающего методы перемещения и включения фар;

с) написать программу, демонстрирующую работу с объектами таких классов.

Прямоугольник (класс **TRectg**) описывается в модуле Unit1. Поля хранят координаты прямоугольника и его цвет. Метод Show рисует прямоугольник на экране, а метод Move перемещает его на заданное расстояние.

Текст заголовочного файла Unit1.h:

```
#ifndef Unit1H
#define Unit1H
//---------------------------------------------------------------------------
#include <graphics.hpp>
class TRectg
{
public:
TRectg(int,int,int,int,TColor,TCanvas*);
void Move(int, int );
virtual void Show(TColor,TColor);
protected:
int   X1,Y1,X2,Y2;
TColor Color;
TCanvas* Canvas;
};
#endif
```

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

Текст файла Unit1.cpp:

```
//---------------------------------------------------------------------------
#pragma hdrstop
#include "Unit1.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)

TRectg :: TRectg(int X01,int Y01,int X02,int Y02,TColor Color0,
                        TCanvas* Canvas0)  // Конструктор
{
X1=X01;
Y1=Y01;
X2=X02;
Y2=Y02;
Color=Color0;
Canvas=Canvas0;
}

void TRectg::Move(int dX, int dY)  // Перемещение на заданное расстояние
{
Show(clBtnFace,clBtnFace);  // Удаление старого изображения
X1+=dX; Y1+=dY;                 // Расчет новых координат
X2+=dX; Y2+=dY;
Show(clBlack,Color);            // Рисование изображение на новом месте
}

void TRectg::Show(TColor CP, TColor CB) // Отображение прямоугольника
{
Canvas->Pen->Color=CP;
Canvas->Brush->Color=CB;
Canvas->Rectangle(X1,Y1,X2,Y2);
}
```

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

Автомобиль (класс TMashin) описывается в модуле Unit2. Метод Move унаследован без изменений, но он обращается к измененному методу Show. Метод SetD предназначен для задания цвета фар.

Текст заголовочного файла Unit2.h:

```
#ifndef Unit2H
#define Unit2H
//---------------------------------------------------------------------------
#endif
#include "Unit1.h"
 class TMashin : public TRectg
 {
 protected:
 TColor ColorD;
 public:
 TMashin(int X01,int Y01,int X02,int Y02,TColor Color0,TCanvas* Canvas0):
TRectg(X01,Y01,X02,Y02,Color0,Canvas0){};
 void SetD(TColor);
 void Show(TColor, TColor);
 };
```

Текст файла Unit2.cpp:

```
        #pragma hdrstop
        #include "Unit2.h"
        //---------------------------------------------------------------------------

        #pragma package(smart_init)


         void TMashin::Show(TColor CP, TColor CB)  // Отображение машины
         {
        Canvas->Pen->Color=CP;
        Canvas->Brush->Color=CB;
        int hx=(X2-X1)/10;
        int hy=(Y2-Y1)/10;
```

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

```
 TPoint pnt[7];
    pnt[0]=Point(X1,Y1+4*hy);
    pnt[1]=Point(X1,Y1+8*hy);
    pnt[2]=Point(X2-2*hx,Y1+8*hy);
    pnt[3]=Point(X2-2*hx,Y1+4*hy);
    pnt[4]=Point(X1+6*hx,Y1+4*hy);
    pnt[5]=Point(X1+4*hx,Y1);
    pnt[6]=Point(X1+2*hx,Y1);
    Canvas->Polygon(pnt,6);  /
 Canvas->Ellipse(X1+hx,Y1+6*hy,X1+3*hx,Y2);
 Canvas->Ellipse(X1+5*hx,Y1+6*hy,X1+7*hx,Y2);
  TPoint pnd[4];
  Canvas->Pen->Color=clBtnFace;
  if (CB != clBtnFace)Canvas->Brush->Color=ColorD;
    pnd[0]=Point(X2-2*hx+1,Y1+5*hy);
    pnd[1]=Point(X2-2*hx+1,Y1+7*hy);
    pnd[2]=Point(X2,Y1+8*hy);
    pnd[3]=Point(X2,Y1+4*hy);
  Canvas->Polygon(pnd,3);  // Ñâåò
}

void TMashin::SetD(TColor clr)  // Задание цвета фар
{
 ColorD = clr;
};
```

Текст файла labOOP.cpp:

```
#include <vcl.h>
#pragma hdrstop

#include "labOOP.h"
#include "unit1.h"
#include "unit2.h"
```

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

```
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
TRectg *Obj1;
TMashin *Obj2;

//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
 // Создание динамических объектов классов
Obj1 = new TRectg(10, 10, 60, 70, clYellow,Form1->Canvas);
Obj2 = new TMashin(100, 10, 240, 70, clRed,Form1->Canvas);
}
//---------------------------------------------------------------------------

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
    // Включение фар
if (CheckBox1->Checked) Obj2->SetD(clWhite);
                        else Obj2->SetD(clBtnFace);
 Obj2->Move(0,0);
}
//---------------------------------------------------------------------------


void __fastcall TForm1::FormResize(TObject *Sender)
{
 // Отображение объектов (прямоугольника и машины)
 Obj1->Move(0,0);
 Obj2->SetD(clBtnFace);
 Obj2->Move(0,0);
}

//---------------------------------------------------------------------------
void Dvigen(int hx, int hy) // Движение выбранного объекта
{
switch (Form1->RadioGroup1->ItemIndex)
{
case 0:
  Obj1->Move(hx,hy);
break;
```

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

```
case 1:
  Obj2->Move(hx,hy);
break;
}
}
//---------------------------------------------------------------------------
void __fastcall TForm1::SpeedButton1Click(TObject *Sender)
{
 Dvigen(0, -StrToInt(Edit2->Text));
}
//---------------------------------------------------------------------------

void __fastcall TForm1::SpeedButton4Click(TObject *Sender)
{
Dvigen(0, StrToInt(Edit2->Text));
}
//---------------------------------------------------------------------------

void __fastcall TForm1::SpeedButton2Click(TObject *Sender)
{
Dvigen(-StrToInt(Edit1->Text), 0);
}
//---------------------------------------------------------------------------

void __fastcall TForm1::SpeedButton3Click(TObject *Sender)
{
Dvigen(StrToInt(Edit1->Text), 0);
}
//---------------------------------------------------------------------------
```

Таким образом, ООП позволяет создавать единые методы для различных, хотя и родственных между собой, объектов. Оно также позволяет постепенно усложнять программу, не изменяя уже созданные классы, а наращивая их путем введения потомков.

9.5. Варианты заданий

Описать класс-родитель и класс-потомок, имеющие методы, указанные в соответствующем варианте задания (потомок наследует или переопределяет методы родителя и приобретает новые). Предусмотреть необходимое количество кнопок для демонстрации каждого из методов объектов.

1. Родитель - круг (перемещение).
   Потомок - колесо (вращение).
2. Родитель - прямоугольник (перемещение).

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

Потомок - повозка (прямоугольник на 2 колесах) (перемещение вперед и назад с поворотом колес).

3. Родитель - отрезок (перемещение в произвольную сторону, поворот вокруг начального конца).

Потомок - ракета (включение/выключение двигателя с появлением/ исчезновением пламени из сопла, смещение вперед).

4. Родитель - эллипс (перемещение).

Потомок - рожица (открывание и закрывание рта, открывание и закрывание глаз).

5. Родитель - неподвижный человечек без головного убора с подвижными, сгибающимися в локтях руками (шевеление руками).

Потомок - солдатик (ввести поле – наличие пилотки) (отдание чести).

6. Родитель - трапеция с горизонтальными основаниями (перемещение).

Потомок - кораблик (смещение вперед, поднятие/спуск флага).

7. Родитель - квадрат (перемещение).

Потомок - квадратная рожица (поворот глаз направо и налево).

8. Родитель - кружок радиусом 3 пиксела (перемещение).

Потомок - выходящая из кружка стрелка (поворот, сдвиг вперед).

9. Родитель - трапеция (перемещение).

Потомок - автомобиль (движение вперед и назад, открывание дверцы).

10. Родитель - неподвижный человечек с подвижными (шевеление руками).

Потомок - сигнальщик (подъем/опускание флажка заданного цвета, выбираемого из нескольких цветов).

11. Родитель - грузовик (смещение вперед/назад).

Потомок - самосвал (ввести поле – наличие груза) (загрузка, откидывание/поднятие кузова).

12. Родитель - домик с дверцей (открывание/закрывание дверцы).

Потомок - домик с дверцей и 2 окнами (открывание/закрывание каждого из окон).

13. Родитель - прямоугольник на 2 колесах (смещение влево/вправо).

Потомок - автофургон (разворот в обратном направлении).

14. Родитель - самолетик (вид сбоку) (перемещение).

Потомок - самолетик с шасси (выпуск/убирание шасси.)

15. Родитель - светофор (смена предыдущего сигнала на очередной (в последовательности: красный -> красный+желтый -> зеленый -> желтый -> красный -> красный+желтый ->...)).

Потомок - светофор со стрелкой поворота (активизация ее с удлинением последовательности (желтый без стрелки -> красный без стрелки -> красный со стрелкой -> красный+желтый без стрелки -> ...); отключение стрелки поворота с возвратом к прежней последовательности).

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

ТЕМА 10. ПРОГРАММИРОВАНИЕ С ИСПОЛЬЗОВАНИЕМ ХЕШИРОВАНИЯ

Цель лабораторной работы: изучить способы программирования алгоритмов с использованием хеширования.

10.1. Понятие хеширования

Имеется глобальная проблема: поиск данных в списке.

1. Если данные расположены беспорядочно в массиве (линейном списке, файле), то осуществляют **линейный поиск** с эффективностью *0(n/2)*.

2. Если имеется упорядоченный массив (двоичное дерево), то возможен **двоичный поиск** с эффективностью $0(\log_2 n)$;

Однако при работе с двоичным деревом и упорядоченным массивом затруднены операции вставки и удаления элементов, дерево разбалансируется, и требуется балансировка. Что можно придумать более эффективное?

Придумали алгоритм **хеширования** (*hashing* - перемешивание), при котором ключи данных записываются в хеш-таблицу. При помощи некой функции *i=h(key)* алгоритм хеширования определяет положение искомого элемента в таблице по значению его ключа.

В простейшем случае алгоритм хэширования реализуется следующим образом.

Допустим, имеется набор m записей с ключами в диапазоне $0 \le key \le K, \quad m < K$.

Создадим массив:

```
<тип записей> H[K];
```

Вначале его очистим **`H[i]:=0`** и наши m записей помещаем в этот массив в соответствии со значением ключа *i=h(key)=key*. Затем используем операторы

```
Zp =H[key]; // Извлекаем из массива
H[key] =zp; // Записываем в массив
```

Хорошо, если ключи располагаются от 1 до 100. А если это номер страховой карточки с 9-значащими цифрами? Потребуется массив из 10^9 ячеек!!! Для одного города с миллионным населением этот массив будет использоваться только на 0,1%.

Чтобы все-таки использовать этот изящный способ, придумывают различные **схемы хеширования**, т.е. установление такой функциональной связи между значением ключа *key* и местоположением *i* записи в некотором массиве, при котором размер таблицы был бы пропорционален количеству записей, а не величине ключа.

Пусть M - предельный размер массива записей $m < M$. Позиции в массиве будем нумеровать начиная с нуля. Задача состоит в том, чтобы подобрать функ-

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

цию *i=h(key)*, которая по возможности равномерно отображает значения ключа *key* на интервал $0 \le i \le M-1$. Чаще всего, если нет информации о вероятности распределения значений ключей по записям, в предположении равновероятности используют *i=h(key)=key % M*.

Функция *h(key)* называется **функцией расстановки,** или **хеш-функцией**. Ввиду того, что число возможных значений ключа *K* обычно значительно превосходит возможное количество записей *K>>M*, любая функция расстановки может для нескольких значений ключа давать одинаковое значение позиции *i* в таблице. В этом случае *i=h(key)* только определяет позицию, начиная с которой нужно искать запись с ключом *key*. Поэтому схема хеширования должна включать **алгоритм разрешения конфликтов**, определяющий порядок действий, если позиция *i=h(key)* оказывается занятой записью с другим ключом.

Имеется множество схем хеширования, различающихся как выбором удачной функции *h(key)*, так и алгоритмом разрешения конфликтов. Эффективность решения реальной практической задачи будет существенно зависеть от выбираемой стратегии.

Алгоритм размещения записей в хеш-таблицу содержит следующие действия: сначала ключ *key* очередной записи отображается на позицию *i=h(key)* таблицы. Если позиция свободна, то в нее размещается элемент с ключом *key*, если занята, то отрабатывается алгоритм разрешения конфликтов, который находит место для размещения данного элемента.

Алгоритм поиска сначала находит по ключу позицию *i* в таблице, и если ключ не совпадает, то последующий поиск осуществляется в соответствии с алгоритмом разрешения конфликтов, начиная с позиции *i*.

Часто ключами таблиц являются не числа, а последовательность букв (строки). При выборе функции *i=H(key)* важно, чтобы ее значения вычислялись, как можно проще и быстрее. Поэтому символьные ключи обычно заменяют целыми числами из некоторого диапазона. Например, если *key* - слово, то сумма кодов первых 2-3-х букв, если фраза (Ф.И.О.), то сумма кодов первых букв. При написании алгоритма разрешения конфликтов поиск ведут уже по полному слову.

10.2. Хеш-таблица на основе массива связанных списков

Один из самых изящных способов разрешения конфликтов состоит в том, чтобы сохранять записи, отображаемые на одну позицию, в связанные списки (обычно стеки). Для этого вводится массив из *M* указателей на связанные списки.

M - простое число, немного большее, чем предполагаемое количество записей.

Хеш-функция выбирается в виде *i:=Key % M*.

Работу с таблицей можно организовать посредством класса THesh, представленного ниже.

Текст заголовочного файла модуля:

```
#ifndef HeshUH
#define HeshUH
```

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

```
#include <vcl.h>
//---------------------------------------------------------------------------
typedef int TKey;
struct TInf {
 AnsiString Inf;
 TKey key;
              };

struct TSel {
 TInf Inf;     // Поле информации
 TSel *A;    // Поле ключа
               };

class THesh   // Класс для работы с таблицей
{
 public:
  THesh(int, char*);   // Конструктор (создание таблицы)
  void Add(TInf);        // Добавление элемента
  void Del(TKey);      // Удаление элемента
  bool Read(TKey,TInf*);  // Чтение  элемента
  ~THesh();              // Деструктор (освобождение таблицы)
 private:
  int M;                 // Количество записей в таблице
  TSel **H;              // Указатель на массив указателей
  char *Filename;    // Имя файла для вывода таблицы
  int i;
};
#endif
```

Текст файла модуля:

```
#pragma hdrstop
#include <stdio.h>
#include "HeshU.h"
```

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

```
#pragma package(smart_init)
//---------------------------------------------------------------------------

THesh::THesh(int k, char *FN)
{
 Filename =FN;
 M = k;
 H = new TSel*[M]; // Создание массива из M указателей
 for (int i=0; i<M; i++) H[i]=NULL; // и его очистка
}
//---------------------------------------------------------------------------

void THesh::Add(TInf Inf)
{
 TSel *p = new TSel;
 i=Inf.key%M;        // Хеш-функция
   p->A = H[i];
   p->Inf = Inf;
   H[i]=p;
}
//---------------------------------------------------------------------------

void THesh::Del(TKey key)
{
 i=key%M;
 TSel *p = H[i], *p1;
  if (p==NULL) return;

  if (p->Inf.key == key)
      {
    H[i]=p->A;
    delete p;
```

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

```
      }
   else
      {
    p1 = p->A;
     while (p1!=NULL){
    if (p1->Inf.key=key) {
        p->A=p1->A;
        delete p1;
         return;
                    }
        p = p1;
        p1 = p1->A;
                }
      }
}
//---------------------------------------------------------------------

bool THesh::Read(TKey key,TInf *Inf)
{
 i=key%M;
 TSel *p=H[i], *p1;
 bool bl=FALSE;
 if (p!=NULL)
  {
   do{
    bl=(p->Inf.key==key);
    p1=p;
    p=p->A;
     } while((! bl)&& (p != NULL));
   if (bl) *Inf=p1->Inf;
 }
 return bl;
```

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

```
}
//---------------------------------------------------------------------------

THesh::~THesh()
{
 TSel *p;
 FILE *Fl;
   if ((Fl=fopen(Filename,"wb"))==NULL)  {
       ShowMessage("File no created");
       return;
                                    }
for (int i=0; i<M; i++) {
  while (H[i]!=NULL) {
   p=H[i];
  fwrite(&H[i],sizeof(TSel),1,Fl);
   H[i]=H[i]->A;
   delete p;
                 }
                  }
 fclose(Fl);
 delete H;
}
```

После описания этого класса работа с хеш-таблицей осуществляется следующим образом:

```
THesh *H1;  // Экземпляр таблицы
...
H1 = new THesh(<кол-во записей>,<имя файла >);  // Создание таблицы
for( i=1; i<N; i++) {
 <ввод Inf>
H1->Add(Inf);  // Добавление элемента
                        }
       ...
<ввод key>
bool bl=H1->Read(key,&Inf);  // Чтение элемента с ключом key
       ...
```

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

```
<ввод key>
H1->Del(key);  // Удаление элемента с ключом key
        ...
H1->~THesh();       // Очистка выделенной памяти
```

Преимущество этого метода заключается в том, что связанные хэш-таблицы никогда не переполняются, довольно просто осуществляется вставка, удаление и поиск элементов. Недостаток таких таблиц в том, что если данные неравномерно перемешаны по ключу, то некоторые стеки могут оказаться очень длинными, в то время как большинство других будут пустыми, при этом поиск будет замедляться.

Для избавления от этого недостатка нужно придумать другую функцию хеширования, используя информацию о распределении записей по значению ключа. Например, можно выбрать двухмерный массив указателей, и подбирая простые числа p и q, $p * q \gtrsim M$, добиваться равномерного распределения значений по таблице.

10.3. Индивидуальные задания

Составить класс для работы с хеш-таблицей на основе массива стеков. В вышеперечисленных методах модифицировать деструктор так, чтобы при обращении к нему происходила запись всех данных из хеш-таблицы в файл с освобождением памяти. Создать обработчик, при обращении к которому все данные из файла записываются в хеш-таблицу. Создать программу записи вводимой информации в хеш-таблицу и поиска требуемой записи по ключу.

В качестве индивидуального задания использовать задания к теме 4.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

ЛИТЕРАТУРА

1. Программирование алгоритмов в среде Builder C++. Лаб. практикум по курсам «Программирование» и «Основы алгоритмизации и программирование» для студентов 1 – 2 курсов всех специальностей БГУИР дневной и вечерней форм обучения. В 2 ч. Ч. 1. /Под общ. ред. А.К. Синицына. – Мн.: БГУИР, 2004.

2. Синицын А.К. Конспект лекций по курсу “Программирование” для студентов 1 – 2 курсов всех специальностей БГУИР. – Мн.: БГУИР, 2001.

3. Вирт Никлаус. Алгоритмы и структуры данных. – СПб.: Невский диалект, 2001.

4. Стивенс Род. Delphi. Готовые алгоритмы. – М.: ДМК Пресс, 2001.

5. Морозов А.А. Структуры данных и алгоритмы. Учеб. пособие. В 2 ч. – Мн.: БГПУ им. М.Танка. Ч. 1. – 2000, Ч. 2. – 2001.

6. Архангельский А.Я. Программирование в C++ Builder 6. – М.: ЗАО “Издательство БИНОМ”, 2002.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

Св. план 2004, поз. 40

Учебное издание

Синицын Анатолий Константинович,
Навроцкий Анатолий Александрович,
Щербаков Александр Владимирович

ПРОГРАММИРОВАНИЕ АЛГОРИТМОВ В СРЕДЕ Builder C++

Лабораторный практикум по курсам “Программирование” и «Основы алгоритмизации и программирование» для студентов 1 – 2-го курсов всех специальностей БГУИР дневной и вечерней форм обучения
В 2-х частях
Часть 2

Библиотека БГУИР

Редактор Е.Н. Батурчик

Подписано в печать 17.02.2005. Формат 60x84 1/16. Бумага офсетная.
Гарнитура «Таймс». Печать ризографическая. Усл. печ. л. 4,77.
Уч.-изд. л. 4,3. Тираж 350 экз. Заказ 670.

Издатель и полиграфическое исполнение: Учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники»
Лицензия на осуществление издательской деятельности №02330/0056964 от 01.04.2004.
Лицензия на осуществление полиграфической деятельности №02330/0133108 от 30.04.2004.
220013, Минск, П. Бровки, 6

PDF created with pdfFactory Pro trial version www.pdffactory.com