Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Кафедра вычислительных методов и программирования

А. К. Синицын, А. А. Навроцкий

ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ В СРЕДЕ DELPHI. АЛГОРИТМЫ НА СТРУКТУРАХ ДАННЫХ

Лабораторный практикум по курсу
«Основы алгоритмизации и программирования»
для студентов 1–2-го курсов всех специальностей БГУИР

Минск 2007

УДК 681.3.06 (075.8)
ББК 32.973-018 я 73
С 38

Рецензент
зав. кафедрой информатики БГУИР, д-р физ.-мат. наук,
проф. Л. И. Минченко

Синицын, А. К.

С 38 Основы алгоритмизации и программирования в среде DELPHI. Алгоритмы на структурах данных : лаб. практикум по курсу «Основы алгоритмизации и программирования» для студ. 1–2-го курсов всех спец. БГУИР / А. К. Синицын, А. А. Навроцкий. – Минск : БГУИР, 2007. – 52 с. : ил.

ISBN 978-985-488-219-2

В лабораторном практикуме приведены краткие теоретические сведения об основных алгоритмах программирования на языке Object Pascal в среде DELPHI. После каждой темы дан набор индивидуальных заданий.

УДК 681.3.06 (075.8)
ББК 32.973-018 я 73

ISBN 978-985-488-219-2

СОДЕРЖАНИЕ

ТЕМА 1. ПРОГРАММИРОВАНИЕ С ИСПОЛЬЗОВАНИЕМ РЕКУРСИИ

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

1.1. 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 – $x_n = \varphi(x_{n-1})$ при $x_0 = a$.

```
Function XR(n:Word):extended;
 begin
   if n=0 then Result:=a
          else Result:=φ(XR(n-1));
 end;
```

Здесь φ – функция, определяющая закон рекуррентности и определяемая в общем случае как

```
Function Fi(x:extended):extended;
begin Result:=(x+b/x)/2; end;
```

или явно, например, как

```
XR:=(XR(n-1)+b/XR(n-1))/2.
```

Обращение: `a:=a0; b:=b0; y:=XR(n0);`

При выполнении вышеприведенной 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.

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.

1.2. Порядок выполнения работы

Задание: Написать отдельный модуль, содержащий описание класса, в состав которого входят методы решения указанной задачи, необходимые поля и свойства класса (**`property`**). Написать основной модуль, содержащий описа-

ние экземпляра объекта класса, ввод исходных данных, обращение к методам класса и вывод результата в удобной форме.

Представить отчет, содержащий распечатку программы, описание рекурсивного (разбиение на тривиальные и элементарные подзадачи) и нерекурсивного алгоритмов задачи. Продемонстрировать работу программы, ответить на контрольные вопросы.

1.2.1. Пример решения задачи

Написать программу вычисления $S = \sum_{i=1}^{n} (i+1)^2 / i$ двумя методами. Один метод вычисляет сумму без использования рекурсии, другой с использованием рекурсии.

Листинг 1.1

```
Unit URecurs;
Interface
 Type
   Tzad1=class(TObject)
      s: extended;
   Function srec(n:word):extended;
   Function snotrec(n:word):extended;
  end;
Implementation

   Function Tzad1.srec;
    begin
     if n=1 then result:=4
            else result:=srec(n-1)+sqr(n+1)/n;
    end;

   Function Tzad1.snotrec;
     Var i:word;
    begin
     s:=0;
        for i:=1 to n do s:=s+sqr(i+1)/i;
     result:=s;
    end;
end. // Конец URecurs;
```

Листинг 1.2

```
Unit Unit1;
        …
 Uses URecurs;
        …
Var zad1:Tzad1;
begin
   …
 zad1:=Tzad1.create;
 x:=zad1.srec(N);
 y:=zad1.snotrec (N);
   < Вывод x, y >
 Zad1.free;
end.
```

1.3. 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 ($p < 10$) в виде строки цифр $a_i, a_{i-1}, \ldots a_0$.

2. В упорядоченном массиве целых чисел a_i, $i=1\ldots n$ найти номер элемента c методом двоичного поиска, используя очевидное соотношение: если $c \le a_{n/2}$, тогда $c \in [a_1 \ldots a_{n/2}]$, иначе $c \in [a_{n/2+1} \ldots a_n]$. Если элемент c отсутствует в массиве, то вывести соответствующее сообщение.

3. Найти наибольший общий делитель чисел M и N. Используйте теорему Эйлера (алгоритм Евклида): 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. Решить задачу о Ханойской башне.

Имеются три стержня: $s1$, $s2$, $s3$. На первом из них нанизаны n дисков различных диаметров, образующих правильную пирамиду – чем выше расположен диск, тем меньше его диаметр. Требуется переместить всю башню на второй стержень, причем диски можно переносить по одному, нельзя помещать диск на диск меньшего диаметра, для промежуточного хранения можно использовать третий диск.

6. Вычислить значение полинома степени n по схеме

$$P_n = \sum_{i=0}^{n} a_i x^i = a_0 + x\left(a_1 + \ldots x\left(a_{n-1} + x a_n\right)\ldots\right).$$

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

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

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; $y=x\cdot x^{N-1}$, если N нечетное.

14. Вычислить $C_n^m=\frac{n!(n-m)!}{m!}$, $m\le n$.

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

$A(0, n) = n+1$;

$A(m,0) = A(m – 1, 1)$; $(m > 0)$;

$A(m, n) = A(m – 1, A(m, n – 1))$; $(m > 0; n > 0)$.

ТЕМА 2. ПРОГРАММИРОВАНИЕ ПЕРЕБОРА ВАРИАНТОВ С ИСПОЛЬЗОВАНИЕМ ДЕРЕВЬЕВ РЕШЕНИЙ

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

2.1. 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 побольше, 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 и проверке всех возможных выборок $\{(i_1,...,i_k),\ 0 \le k \le n\}$ из n значений $\{1, 2, ..., n\}$. Например, для $n = 3$ таких выборок $2^n = 8$: $\{1, 2, 3\}$, $\{1, 2\}$, $\{1, 3\}$, $\{2, 3\}$, $\{1\}$, $\{2\}$, $\{3\}$, $\{\ \}$.

2.2. Рекурсивная процедура метода ветвей и границ

Стратегия отсечения заведомо неприемлемых и неоптимальных решений позволяет резко сократить количество просматриваемых вариантов.

Ниже приведен листинг 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 `Sopt` из индексов, входящих в выборку элементов. В процедуре `VbrVG(i,tw,oct)` введены дополнительно два формальных параметра: `tw` – суммарный вес текущей выборки и `oct` – общая ее стоимость, 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 `oct` будет не меньше полученной до этого стоимости C_{max} оптимальной выборки, находящейся в `Sopt`:

```
oct-a[i].c > Cmax.
```

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

2.3. 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 представлены далее.

2.3.1. 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*.

2.3.2. 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.

2.3.3. 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.

2.3.4. Метод случайного поиска

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

2.4. Порядок выполнения работы

Задание: Написать отдельный модуль, содержащий класс, в состав которого входят пять методов решения указанной задачи и необходимые свойства класса (property). Составить программу решения задачи оптимального выбора различными методами.

2.4.1. Пример решения задачи

Листинг 2.1

```
Unit UPerebor;
Interface
 Type
  TElem=Record
   c,w : Extended
        end;
  TZad2=class(TObject)
    n : byte;
    a : array[1..255] of TElem;
    S, Sopt : set of byte;
    Wmax, Cmax, oct1, wt, ct: Extended;
   Procedure VbrPP(i:byte);
   Procedure VbrVG(i:byte; tw,oct:Extended);
     < Добавить сюда необходимые методы >
     …
 End;
 Implementation

Procedure Tzad2.VbrPP(i:byte);
 var j:byte;
begin
 for j:=0 to 1 do begin
  if j=0 then begin wt:=wt+a[i].w; ct:=ct+a[i].c;
                    Include(s,i) end
          else begin wt:=wt-a[i].w; ct:=ct-a[i].c;
                   Exclude(S,i); end;
  if i<n then   VbrPPj(i+1)
         else if (wt<=Wmax) and (ct>Cmax) then
                        begin Cmax:=ct;Sm:=S; end;
                  end;//j
end;
```

```
Procedure TZad2.VbrVG;
 begin
          // Попытка включения
   if tw+a[i].w<=Wmax then begin
     Include(s,i);
   if i<n then VbrVG(i+1,tw+a[i].w,oct)
          else
     if oct>Cmax then begin
               Cmax:=oct;
               Sopt:=S
                        end;
      Exclude(S,i);
                              end;
        //Попытка исключения
   oct1:=oct-a[i].c;
   if oct1>Cmax then
    if i<n then VbrVG(i+1,tw,oct1)
           else begin
             Cmax:=oct1;
            Sopt:=S
               end;
 End; // Конец процедуры VbrVG
                …
    < Добавить сюда необходимые методы >
                …
End. // Конец модуля UPerebor;

Unit unit1;
                …
  Uses UPerebor; // Подключение модуля UPerebor;
 Var
   zad2 : TZad2;
   ocm  : extended;
                …
Procedure TForm1.FormCreate(Sender: TObject);
begin
 Edit1.Text:='3';
 Edit2.Text:='10';
 StringGrid1.Cells[1,0]:='вес';
 StringGrid1.Cells[2,0]:='цена';
 for i:=1 to 10 do
    StringGrid1.Cells[0,i]:=IntToStr(i);
 for i:=1 to 3 do begin
```

```
    StringGrid1.Cells[1,i]:=IntToStr(i*i);
    StringGrid1.Cells[2,i]:=IntToStr(i+5);
                  end;
 Memo1.Clear;
end;

Procedure TForm1.Button1Click(Sender: TObject);
Begin
 Memo1.Clear;
zad2:=Tzad2.create;
  With zad2 do begin
 n:=strToInt(Edit1.Text); ocm:=0;
  for i:=1 to n do begin
    a[i].w:=StrToInt(StringGrid1.Cells[1,i]);
    a[i].c:=StrToInt(StringGrid1.Cells[2,i]);
    ocm:=ocm+a[i].c;
                  end;
   Wmax:=StrToInt(Edit2.Text);
   Cmax:=0; S:=[]; Sopt:=[];
   VbrVG(1,0,ocm);
 Memo1.Lines.Add('Оптимальный вариант');
 Memo1.Lines.Add(' i      w      c');
  for i:=1 to n do
    if i in Sopt then
   Memo1.Lines.Add(IntToStr(i)+'  '+
            FloatToStrF(a[i].w,ffFixed,8,3)+'  '+
            FloatToStrF (a[i].c,ffFixed,8,3));
Memo1.Lines.Add('Cmax='+
 FloatToStrF (Cmax,ffFixed,8,3)+
              '  Wmax='+
 FloatToStrF (Wmax,ffFixed,8,3));
                    End;
  Zad2.Free;
end;
```

Панель диалога будет иметь вид, показанный на рис. 2.1.

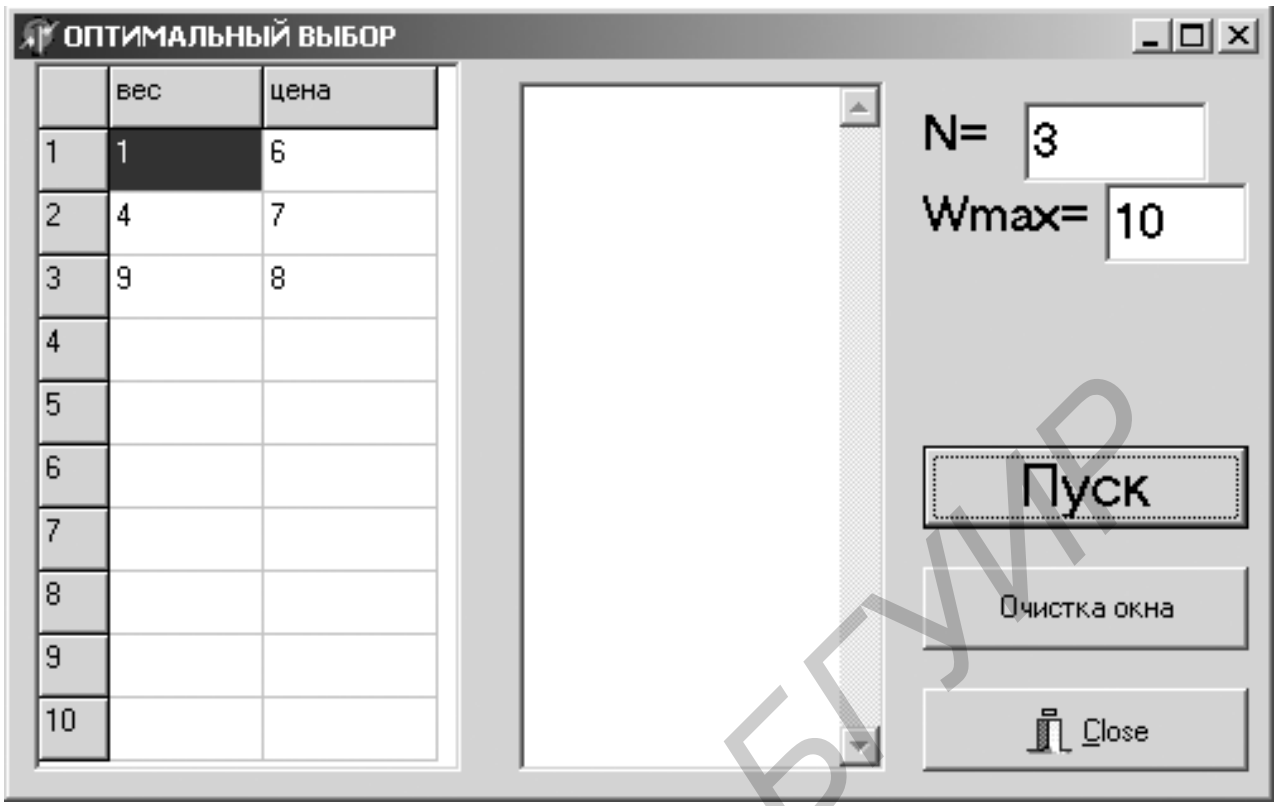

Рис. 2.1

2.5. Варианты задач

Задана таблица из 5 элементов:

Вес	$1+N_v$	11	12	13	14
Цена	18	20	17	19	$28-N_v$

Здесь N_v – номер варианта 1–15.

Для всех вариантов выполнить:

1. Для двух значений W_{max} = 30 и W_{max} = 40 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).

Использовать компонент `RadioGroup`.

Получить решение и сравнить время выполнения для n = 3, 5, 10, 20.

ТЕМА 3. ПОИСК И СОРТИРОВКА МАССИВОВ

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

3.1. Организация работы с базами данных

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

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

1. Вносятся изменения и осуществляется поиск прямо на диске с использованием специфической техники работы с записями на файлах, при этом временные затраты на обработку данных (поиск, сортировку) значительно возрастают, но нет ограничений на оперативную память.

2. В начале работы вся база (или ее необходимая часть) считывается в массивы записей и обработка производится в оперативной памяти, что значительно сокращает время, однако требует затрат оперативной памяти.

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

Обычно запись содержит некое ключевое слово (ключ), по которому ее находят среди множества других аналогичных записей. В вышеприведенном примере в зависимости от решаемой задачи ключом может служить фамилия или номер группы 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.

3.2. Поиск в массиве записей

Задача поиска требуемого элемента в массиве записей $a[i]$, $i = 1...n$ заключается в нахождении индекса i, удовлетворяющего условию $a[i].k = x$. Здесь ключ k выделен в отдельное поле, x – аргумент поиска того же типа, что и k. После нахождения i обеспечивается доступ ко всем другим полям найденной записи $a[i]$.

3.2.1. 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.

3.2.2. Поиск делением пополам

Используется, когда данные упорядочены, например, по возрастанию ключа k, т.е. $a[i].k \le a[i+1].k$. *Основная идея* – берем «средний» (m) элемент. Если $a[m].k < x$, то все элементы $i \le m$ можно исключить из дальнейшего поиска, если $a[m].k \ge x$, то можно исключить все $i > m$.

3.3. Сортировка массивов

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

Методы 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 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 $n \ge 100$.

Среди простых методов наиболее популярны:

1) метод прямого обмена;

2) метод прямого выбора.

Реже используются:

3) сортировка с помощью прямого (двоичного) включения;

4) **шейкерная** сортировка (модификация пузырьковой).

Улучшенные методы сортировки:

1) **метод Д. Шелла**, усовершенствование метода прямого включения;

2) сортировка с помощью дерева, метод **HeapSort** (Д. Уильямсон);

3) сортировка с помощью разделения, метод **QuickSort** (Ч. Хоар), улучшенная версия пузырьковой сортировки. На сегодняшний день это самый эффективный метод сортировки.

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, чем метод Шелла.

Идея метода разделения **QuickSort** в следующем.

Выберем значение ключа среднего m-го элемента $x = a[m].k$. Будем просматривать массив слева до тех пор, пока не обнаружим элемент $a[i].k > x$. После этого будем просматривать массив справа, пока не обнаружим $a[j].k < x$. Поменяем местами элементы $a[i]$ и $a[j]$ и продолжим такой процесс просмотра (слева и справа, обмен), пока оба просмотра не встретятся где-то внутри массива. В результате массив окажется разбитым на левую часть $a[1]$, $1 \le l \le j$ с ключами меньше (или равными) x и правую $a[p]$, $i \le p \le n$ с ключами больше (или равными) x. 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.

3.4. Порядок выполнения работы

Задание: Написать программу работы с файлами с использованием класса. Класс содержит следующие стандартные методы: включение новой записи в файл, чтение файла в массив, запись массива в файл, линейный поиск в массиве, сортировка массива методами слияния и `QuickSort`, двоичный поиск в отсортированном массиве, удаление записи из массива ***по заданному ключу***.

Для выполнения индивидуального задания дополнить класс «своим» методом, который решает задачу выбранного варианта.

3.4.1. Пример фрагмента программы

Листинг 3.1

```
Unit UZapSort;
  Interface
  const Mr=1000;        // Максимальная размерность массива
 Type
            …
  Tzp = Record
      zp : TInf;              // Поле информации
      k  : Tk;                // Поле ключа
         end;
  Mas=array[1..Mr] of Tzp;     // Тип – массив записей
   TZad3=class(TObject)
     a : mas;
      n : Word; // Текущий размер массива ≤Mr
   Procedure ReadFromFile;
  Procedure WriteTofile;
  Procedure WriteToMemo;
  Procedure PoiskL;
  Procedure PoiskD;
  Procedure SortSlip;
  Procedure Remov;
   Procedure QuickSort;
 … < Поля и методы, которые надо дописать >
         End;
  Implementation
 Procedure Tzad3.QuickSort;
  … < Описание других методов >
   const M=12;
  Var i,j,L,R : Word; x : Tk; w : Tzp;
      S : 0..M;
```

```
  Stak :array[1..M] of Record L,R:word end;
begin
     s:=1; stak[1].L:=1; stak[1].R:=n;
Repeat       // Выбор из Stak последнего запроса
     L:=stak[s].L; R:=stak[s].R; s:=s-1;
  Repeat     // Разделение a[L] … a[R]
     i:=L; j:=R; x:=a[(L+R) div 2].k;
   Repeat
  While a[i].k<x do i:=i+1;
  While x<a[j].k do j:=j-1;
  if i<=j then begin
  w:=a[i]; a[i]:=a[j]; a[j]:=w;
 i:=i+1; j:=j-1 end;
   until i>j;
  if j-L<R-i then begin      // Какая половина длиннее?
  if i<R then  // Запись запроса из правой части
   begin s:=s+1; stak[s].L:=i; stak[s].R:=R; end;
    R:=j;
   end       // Теперь L и R ограничивают левую часть
       else begin
  if L<j then  // Запись запроса из левой части
   begin s:=s+1;stak[s].L:=L;stak[s].R:=j end;
   L:=i end;
  until L>=R;      // Конец разделения очередной части
until s=0;  // Stak пуст
END;        // Конец метода QuickSort
End.
```

Листинг 3.2

```
Unit Unit1;
…
 Uses UZapSort;
Var zad3:Tzad3;
…
begin
   zad3:=Tzad3.create;
   zad3. ReadFromFile;  // При чтении определяется n
   zad3. QuickSort;
   zad3. WriteTofile;
       < Вывод сортированного массива >
   zad3.free;

end.
```

3.5. Индивидуальные задания

1. В магазине формируется список лиц, записавшихся на покупку товара. Каждая запись этого списка содержит: порядковый номер, Ф.И.О., домашний адрес покупателя и дату постановки на учет. Ключ: дата постановки на учет.

Удалить из списка все повторные записи, проверяя Ф.И.О.

2. Список товаров, имеющихся на складе, включает в себя наименование товара, количество единиц товара, цену единицы и дату поступления товара на склад. Ключ: наименование товара.

Вывести в алфавитном порядке список товаров, хранящихся больше месяца, стоимость которых превышает 1 000 000 р.

3. Для получения места в общежитии формируется список студентов, который включает Ф.И.О. студента, группу, средний балл, доход на члена семьи. Ключ: 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.

4. В справочной автовокзала хранится расписание движения автобусов. Для каждого рейса указаны его номер, тип автобуса, пункт назначения, время отправления и прибытия. Ключ: время отправления.

Вывести информацию о рейсах, которыми можно воспользоваться для прибытия в пункт назначения раньше заданного времени.

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. Ключ: время разговоров.

Вывести по каждому городу: общее время разговоров с ним и сумму.

6. Информация о сотрудниках фирмы включает: Ф.И.О., табельный номер, количество проработанных часов за месяц, почасовой тариф. Ключ: размер заработной платы.

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.

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

Вывести информацию о самой молодой команде.

8. Для книг, хранящихся в библиотеке, задаются: регистрационный номер книги, автор, название, год издания, издательство. Ключ: автор.

Вывести список книг с фамилиями авторов в алфавитном порядке, изданных после заданного года.

9. Различные цеха завода выпускают продукцию нескольких наименований. Сведения о выпущенной продукции включают: наименование, количество, номер цеха. Ключ: количество выпущенных изделий.

Для заданного цеха необходимо вывести количество выпущенных изделий по каждому наименованию в порядке убывания количества.

10. Информация о сотрудниках предприятия включает: Ф.И.О., номер отдела, должность, дату начала работы. Ключ: дата начала работы.

Вывести списки сотрудников по отделам в порядке убывания стажа.

11. Ведомость абитуриентов, сдавших вступительные экзамены в университет, содержит: Ф.И.О., адрес, оценки. Ключ: Ф.И.О.

Вывести в алфавитном порядке фамилии абитуриентов, проживающих в г. Минске и сдавших экзамены со средним баллом не ниже 4,5.

12. В справочной аэропорта для каждого рейса указаны: номер рейса, тип самолета, пункт назначения, время вылета. Ключ: пункт назначения.

Вывести все номера рейсов, типы самолетов и время вылета для заданного пункта назначения в порядке возрастания времени вылета.

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. Ключ: 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.

14. Ведомость абитуриентов, сдавших вступительные экзамены в университет, содержит: Ф.И.О. абитуриента, оценки. Ключ: Ф.И.О.

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.

15. В радиоателье хранятся квитанции о сданной в ремонт радиоаппаратуре. Каждая квитанция содержит следующую информацию: наименование группы изделий (телевизор, радиоприемник и т. п.), марку изделия, дату приемки в ремонт, состояние готовности заказа (выполнен, не выполнен). Ключ: дата приемки в ремонт.

Вывести информацию о состоянии заказов на текущие сутки по группам изделий.

ТЕМА 4. РАБОТА СО СПИСКАМИ НА ОСНОВЕ ДИНАМИЧЕСКИХ МАССИВОВ

Цель лабораторной работы: изучить способы работы с динамическими массивами данных и организацию работы со списками, используя указатели.

4.1. Работа со списками

Список – это группа объектов (данных), с которыми надо работать в оперативной памяти: добавлять в группу новые объекты, удалять используемые, сортировать, находить нужные. В процессе работы список может возрастать и уменьшаться.

Простейшая форма организации списка – это массив данных. Данные, размещенные в массиве, имеют свой номер (индекс), и это придает эффект «видимости» каждому элементу. Мы уже научились некоторым приемам работы с массивами и оценили их эффективность.

Рассмотрим, как организуется работа со списками на основе массива переменного размера.

Предположим, что мы создали начальный список в массиве a размером n с помощью вышеописанного фрагмента программы.

При добавлении нового элемента в список необходимо выделить память на один элемент большую, затем скопировать старый список в новый раздел памяти, добавить туда еще один элемент, после чего освободить ранее выделенную память и установить указатель a на новый раздел памяти.

4.2. Порядок выполнения работы

Для организации работы со списком на основе динамического массива напишем класс `TListm`, содержащий метод добавления элемента в конец списка и метод чтения и удаления элемента из начала списка:

Листинг 4.1

```
Unit Spisok;
Interface
Type  TInf = < тип элементов >;
      TMs = array[1..1] of TInf;
      Pms=^TMs; // Указатель на массив
  TListm=Class(TObject)
     a,a1 : pms;  n,mt:word;
 constructor Create;
 Procedure addk(c:TInf); // Добавить элемент
 Procedure Read1(Var c:Tinf); //Удалить элемент
         end;
```

```
Implementation
constructor TListm.create;
 begin
   Inherited create;
     mt:= sizeof(TInf);  n:=0; a:=Nil;
 end;
Procedure TListm.Addk;
 var i:word;
 begin
  GetMem(a1,(n+1)* mt);
   a1[n+1]:=c; // Добавление нового элемента
 if n>0 then begin
 for i:=1 to n do a1^[i]:=a^[i];
  FreeMem(a,mt*n);
             end;
  a:=a1; n:=n+1;
 end;
 Procedure TListm.Read1;
   var i:word;
 begin
   if n>0 then begin
      c:=a[1]; n:=n-1; // Чтение первого элемента
        if n>0 then begin
                 GetMem(a1,n* mt);
      for i:=1 to n do a1^[i]:=a^[i+1];
                      end
                 else a1:=nil;
                 FreeMem(a,mt*(n+1)); a:=a1;
                end;
   end;  End.
```

4.3. Индивидуальные задания

Используя варианты типов записей из подразд. 3.5, организовать работу со списками записей на основе динамического массива.

1. Составить класс, аналогичный разработанному в предыдущей работе, используя вместо статического массива список на основе динамического массива `TList`, имеющий методы добавления элемента в конец списка и чтения и удаления из начала списка. *Класс* `TList` *сделать родительским*.

2. Включить методы записи списка в файл и чтение списка из файла.

3. Составить методы сортировки списка по ключу, модифицируя и используя методы из темы 3.

4. Начальный список прочесть из файла.

5. Полученные результаты вывести в окно `TMemo`.

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

Цель лабораторной работы: получить навыки программирования однонаправленных списков в виде стека.

5.1. Основные понятия и определения

Стек – это структура данных, организованная по принципу «последний вошел – первый вышел». Образно это можно представить, как запаянную с одной стороны трубку, в которую закатываются шарики. Первый закаченный 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 `sp1` (*вершиной стека*), при этом вводится следующий *рекурсивный* тип, определяющий ячейку памяти *с косвенной адресацией*:

```
Type TSel=^Sel; // Указатель на запись
   Sel=Record // Объявление записи
  inf:TInf;   // Поле информации
   A : TSel; // Указатель на предыдущую ячейку стека
       End;
```

5.2. Порядок выполнения работы

Для организации работы со списком в виде стека на основе косвенной адресации напишем класс `TLists`, содержащий методы добавления и удаления элемента из начала списка:

Листинг 5.1

```
Unit UStec;
Interface
Type
   TInf =  < тип элементов >;
  TSel=^Sel;
  Sel=Record
    inf:TInf;
    A : TSel;
       End;
TLists=class(TObject)
  sp1,spk,sp : TSel;
   constructor create;
```

```
    Procedure Adds (Inf:Tinf);
    Procedure Reads (Var Inf:Tinf);
        …    // Другие методы
 end;
Implementation

constructor TLists.create;
begin
    inherited create;
    sp1:=nil;
end;

 Procedure TLists.Adds;
   begin
   New(sp);
   sp^.Inf:=Inf;
   sp^.A:=sp1;
   sp1:=sp;
   end;

 Procedure TLists.Reads; // Перед чтением необходимо
   Begin                 // убедиться, что стек не пуст
     Inf:=sp1^.Inf;
     sp:=sp1;
     sp1:=sp1^.A;
     Dispose(sp);
   end;

…    // Другие методы
End.
```

Для работы со стеком целесообразно дополнить класс разнообразными методами.

5.3. Индивидуальные задания

Создать класс (в Unit2), реализующий стандартные методы работы со стеком Adds, Reads, Print, ReadAfter, AddAfter, Poisk, PoiskAfter, SortBublInf, SortBublAfter. Написать программу (Unit1), иллюстрирующую работу всех методов работы со стеком. Результат формирования и преобразования стека показывать в компонентах TListBox. После этого на базе стандартного родительского класса написать свой класс, реализующий метод решения своего варианта. Написать обработчик события, реализующий вызов метода решения своего варианта.

1. Создать стек со случайными целыми числами в диапазоне от –50 до +50 и преобразовать его в два стека. Первый должен содержать только положительные числа, а второй – отрицательные. Порядок чисел должен быть сохранен, как в первом стеке.

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

3. Создать стек из случайных целых чисел 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. Подсчитать, сколько элементов стека, построенного из случайных чисел, превышает среднее значение от всех элементов стека.

6. Создать стек из случайных целых чисел и найти в нем максимальное и минимальное значение.

7. Создать стек из случайных целых чисел и определить, сколько элементов стека, начиная с вершины, находится до элемента с максимальным значением.

8. Создать стек из случайных целых чисел и определить, сколько элементов стека, начиная от вершины, находится до элемента с минимальным значением.

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

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

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 и определить наиболее часто встречающееся число.

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

15. Создать стек из случайных целых чисел и получить из него транспонированный.

16. 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 .....».

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

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

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

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

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

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

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

24. 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.

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

26. 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.

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

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

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

30. 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.

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

Цель лабораторной работы: получить навыки программирования списков, организованных в виде очереди.

6.1. Основные понятия и определения

Очередь на основе однонаправленного списка 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 другого конца извлекаем.

Для элемента очереди вводится такой же рекурсивный тип, как и для стека. Отличается организация работы с очередью от работы со стеком только тем, что в списке поддерживается две точки входа – начальная `sp1` и конечная `spk`. При этом добавляются элементы в конец очереди (`Addk`), а извлекаются из начала (`Read1`).

Циклические связанные списки используются, когда нужно обходить набор элементов в бесконечном цикле. В этом случае отсутствуют понятия начала и конца списка.

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 рекурсивный тип с двумя адресными ячейками.

6.2. Порядок выполнения работы

Для организации работы со списком в виде очереди на основе косвенной адресации напишем класс `TListo` как потомок от `TLists`, содержащий методы добавления и удаления элемента в очередь с поддержкой двух точек доступа.

Листинг 6.1

```
Unit Ochered;
Interface
 uses Ustec; // Модуль, содержащий класс TLists
Type
TListo=class(TLists)
   constructor create;
   Procedure Addk (Inf:TInf);
       …  //Другие методы
```

```
        end;
Implementation

constructor TListo.create;
 begin
  inherited create;
   spk:=nil;
 end;

 Procedure TListo.Addk(Inf:TInf);
   begin
   if spk=Nil then begin // Если список пуст
        New(spk);
        spk^.A:=Nil;
        spk^.Inf:=inf;
        sp1:=spk;
                    end
               else begin
   New(spk^.A); // Создание нового элемента
                // за последней записью
   spk:=spk^.A; // Адрес последнего элемента
   spk^.Inf:=Inf;
   spk^.A:=Nil;
                    end;
 end;

   …          // Другие методы

end.
```

6.3. Индивидуальные задания

Создать модуль UStec, содержащий класс, реализующий стандартные методы работы с очередью Addk(Inf), Reads, Print, ReadAfter, AddAfter, Poisk, PoiskAfter, процедуру SortSlip. Написать программу (Unit1), иллюстрирующую работу всех методов работы с очередью. Результат формирования и преобразования очереди показывать в компонентах TListBox. После этого на базе стандартного родительского класса написать свой класс, реализующий метод решения своего варианта. Написать обработчик события, реализующий вызов метода решения своего варианта.

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.

6. Создать двухсвязанный список из случайных целых чисел. Удалить все отрицательные элементы списка.

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 списка, 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 `TMemo` 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 `TImage`.

14. Создать двухсвязанный список из случайных чисел. Преобразовать его в кольцо. Предусмотреть возможность движения по кольцу в обе стороны с отображением места положения текущего элемента с помощью компоненты `TGauge(gkPie)` и числового значения – с помощью `TLabel`.

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.

ТЕМА 7. ИСПОЛЬЗОВАНИЕ СТЕКА ДЛЯ ПРОГРАММИРОВАНИЯ АЛГОРИТМА ВЫЧИСЛЕНИЯ АЛГЕБРАИЧЕСКИХ ВЫРАЖЕНИЙ

Цель лабораторной работы: получить навыки 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 (ОПЗ)**. 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.

Решение задач преобразования инфиксного выражения в постфиксную запись и вычисление выражения в ОПЗ оформить в виде класса с соответствующими методами. Использовать ранее описанный класс `TLists` работы с однонаправленным списком, в котором поле `Inf` имеет тип `Char` (см. тему 5).

7.2. Порядок написания программы

Задание: Написать программу расшифровки и вычисления арифметических выражений с использованием стека. 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. Для контроля вычислить выражение обычным образом.

Листинг 7.1

```
unit UArVr;
  Interface
Uses UStec, math;
Type
Tpz=class(TObject)
 mszn : array['a'..'я'] of extended; // [a..z..я]
  stec:TLists;
Function OBP(stri:string): string;
Function AV(strp:string) : extended;      End;
  Implementation
 var i:byte;
Function Tpz.OBP; // Преобразование
 Var pc : 0..3;  ch,ch1 : char;
Function prior(ch:char):byte;  // Приоритет
 begin
    case ch of
  '(', ')': Result:=0;
  '+', '-': Result:=1;
  '*', '/': Result:=2;
    '^'   : Result:=3;
    end;
  end;
 begin // Начало функции OBP
Result:='';
stec:=TLists.create;
 for i:=1 to length(stri) do begin
   ch:=stri[i];
if not (ch in ['+','-','*','/','(',')','^'])
then Result:=Result+ch //Записываем в вых. строку
else
 if stec.sp1=Nil then stec.Adds(ch)  // Стек пуст
                 else
  if ch='(' then stec.Adds(ch) // Открытая скобка
            else
   if ch=')' then begin // Закрытая скобка
     stec.Reads(ch); // Выталкивание всех операций
     While ch<>'(' do  begin    // до ближайшей '('
      Result:=Result+ch;
      stec.Reads(ch);
                        End
                   End
    else begin // Выталкивание более приоритетных
```

```
        pc:=prior(ch);
       While (stec.sp1<>Nil) and
             (pc<= prior(stec.sp1.Inf)) do begin
                 stec.Reads(ch1);
                 Result:=Result+ch1;
                                              end;
     stec.Adds(ch); // Добавить после выталкивания
    end;
   end;
While stec.sp1<>nil do begin
         stec.Reads(ch);
         Result:=Result+ch
                          end;
  stec.Free;
end; // Конец функции ОВР

Function Tpz.AV;   // Вычисление
  Var ch,ch1,ch2,chr : char;
      op1,op2 : extended;
begin
   stec:=TLists.create;
     chr:=Succ('z'); // Следующий за 'z' символ
   for i:=1 to length(strp) do begin
     ch:=strp[i];
   if not (ch in ['*','/','+','-','^'])
     then stec.Adds(ch)
     else begin
       stec.Reads(ch2); stec.Reads(ch1);
       op1:=mszn[ch1];  op2:=mszn[ch2];
       case ch of
        '+': Result:=op1+op2;
        '-': Result:=op1-op2;
        '*': Result:=op1*op2;
        '/': Result:=op1/op2;
        '^': Result:=power(op1,op2);
       end;
        mszn[chr]:= Result;
        stec.Adds(chr);
        Inc(chr);
           end;
                                  end;
  stec.Free;
 end;
end.
```

Пример вычисления выражения с использованием класса:

```
Var pz:Tpz;
  Begin
   pz:=Tpz.create;
    Edit2.text:=pz.OBP(Edit1.Text);
     with pz,StringGrid1 do begin
  mszn[Cells[0,1][1]]:=StrToFloat(Cells[1,1]);
  mszn[Cells[0,2][1]]:=StrToFloat(Cells[1,2]);
  mszn[Cells[0,3][1]]:=StrToFloat(Cells[1,3]);
  mszn[Cells[0,4][1]]:=StrToFloat(Cells[1,4]);
  mszn[Cells[0,5][1]]:=StrToFloat(Cells[1,5]);
                                  end;
    Edit3.text:=FloatToStr(pz.AV(Edit2.Text));
   pz.free;
  end;
```

Панель диалога должна иметь вид, показанный на рис. 7.1.
Массив `mszn` считать из `StringGrid`.

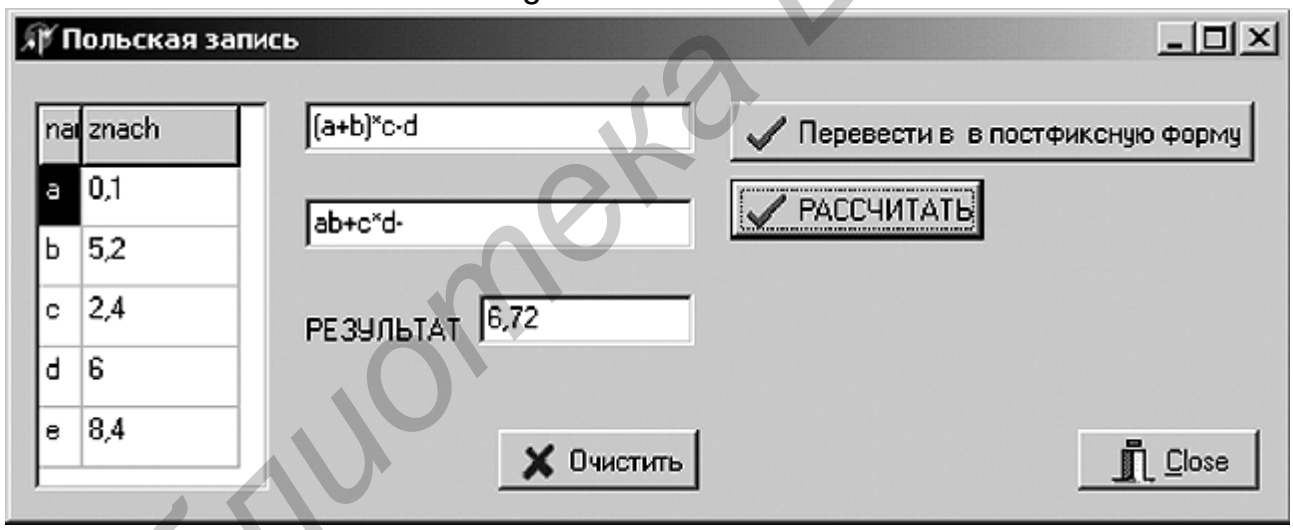

Рис. 7.1

7.3. Индивидуальные задания

1. r = x^(y+z*s)*d-c/s;
2. r = a-b^d*(e+s/d)-w;
3. r = d^(u+v/c)*(s-w);
4. r = b*((x-y/c)*(z^d+a))-e;
5. r = e+s*(a-t^e)+q/p;
6. r = (y-z/v)*a^c/d+u;
7. r = ((t+v-a/b)+(s-z^d))*k;
8. r = (d-c/x^e)*z+f*e;
9. r = a+c-e^(x-y/z)*s;
10. r = s*(a^x+b^y)-z/f;
11. r = (d-e)^f*g+s/a+w;
12. r = t*(y^(z-u)+f)-h/m;
13. r = g*(u+j*n)^s+a/p;
14. r = f-t^(k*(s-h)-n*d^a);
15. r = k*(m+n*j)^i-v/u;

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

Цель лабораторной работы: получить навыки 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.

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

```
Type Ptree=^tree
     tree=Record
      Inf : PInf;
       A1 : Ptree;     // A1 ... Am – указатели на адреса,
        …              //  по которым расположены сестры
       Am : Ptree;     // Если сестра отсутствует, то
           end;        // соответствующий адрес равен Nil
```

После того как дерево заполнено информацией, его нужно уметь просмотреть, распечатать, осуществить поиск. Для работы с деревьями используют набор специфических алгоритмов.

8.2. Компонент TTreeView

Компонент `TTreeView` из меню компонентов `Win32` 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`.

`Function` `AddFirst(Node:TtreeNode;`**`const`** `S:String) : TtreeNode; -` 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.

`Function` `AddChild(Node:TTreeNode;` **`const`** `S:String) : TTreeNode -` добавляет очередной дочерний узел к узлу `Node`.

`Function` `AddChildFirst(Node:TTreeNode;` **`const`** `S:String) : TTreeNode; -` добавляет новый узел как первый из дочерних узлов узла Node.

Procedure `Clear;` - очищает список узлов, описанных в свойстве Items.

Function `FullExpand;` - раскрывает все узлы при отображении дерева.

8.3. Бинарное дерево поиска

Наиболее часто для работы со списками используется бинарное (имеющее степень 2) дерево поиска, в котором ключи расположены таким образом, что для любого узла значения ключа у левого преемника меньше, чем у правого. Таким образом, организованное дерево получило название **бинарное 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 $O(n/2)$.

При организации списков в виде двоичного дерева необходим пакет программ, реализующих следующие действия: поиск заданного ключа; поиск минимального (максимального) ключа; вставка нового значения ключа, не изменяя свойств дерева поиска; удаление заданного ключа; формирование дерева поиска; балансировка дерева.

8.4. Порядок написания программы

Задание. В качестве примера рассмотрим проект, который создает дерево, отображает его в компонентах `TTreeView` и `TMemo` и удаляет дерево. Панель диалога будет иметь вид, представленный на рис. 8.1.

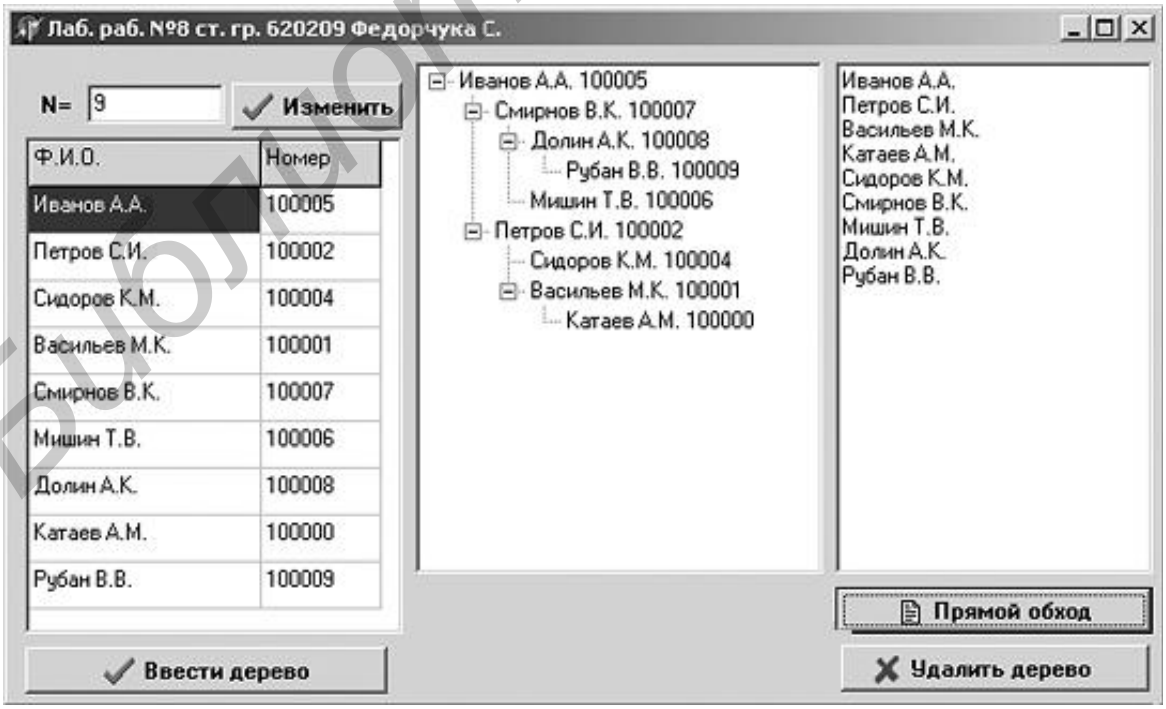

Рис. 8.1

Элементы текста программы приведены ниже.

Листинг 8.1

```
Unit UDerevo;
interface
uses SysUtils;
 Type
   Tkey = LongWord;
   Tinf = Record
       S   : string[80];      // Ф.И.О.
       key : Tkey;            // Номер паспорта
         end;
   TMs=array[1..10] of TInf;
 Ptree = ^tree;
 Tree = Record
      Inf:TInf;
      A1:Ptree;
      A2:Ptree;
        end;
 TtreeB=Class(TObject)
  proot,p,q,v,w : Ptree;
  Constructor create;
  Procedure Add(Inf:TInf);
  Procedure Make1(a:Tms; n:word);
  Procedure View;
  Procedure Write;
  Destructor Free;
      End;
Implementation

 uses unit1;

Constructor TtreeB.create;
 Begin
  Inherited create;
   proot:=nil;
 End;

Procedure TtreeB.Add(Inf:TInf); // Добавление
  Var bl:Boolean;                // нового листа
  begin
      New(w);     // Создание нового листа:
   w^.Inf:=Inf;
   w^.A1:=Nil;
   w^.A2:=Nil;
     if proot=Nil then proot:=w
```

```
                    else begin      // Поиск места для
                   p:=proot;        // добавления листа
    repeat
     q:=p;          // q – указатель на предыдущий элемент
      bl:=Inf.key<p^.Inf.key;
       if bl then p:=p^.A1
             else p:=p^.A2;
    until p=Nil;                // Добавление листа
            if bl then q^.A1:=w
                  else q^.A2:=w;
                       end;
end;

Procedure TtreeB.Make1(a:Tms; n:word); // Создание
 var i:integer;                        // дерева
   begin
  For i:=1 to n do Add(a[i]);
 end;

Procedure TtreeB.View;      // Отображение дерева
   var kl : integer;
 Procedure VW(p:ptree;Var kl:Integer);
   Begin
    if  p <> Nil then
        with Form1.TreeView1.Items do begin
    if kl=-1 then
AddFirst(Nil, p^.Inf.s+' '+IntToStr(p^.Inf.key))
           else
AddChildFirst(Form1.TreeView1.Items[kl],
              p^.Inf.s+' '+IntToStr(p^.Inf.key));
      Inc(kl);
      VW(p^.A1,kl);
      VW(p^.A2,kl);
      Dec(kl);
                  end;
      end;
begin
  Form1.TreeView1.Items.Clear;
  p:=proot;
    kl:=-1;
   VW(p,kl);
  Form1.TreeView1.FullExpand;
end;
```

```
Destructor TtreeB.Free;    // Удаление дерева
 Procedure Delett(p:Ptree);
  Begin
   If p=nil then Exit;
      Delett(p^.A1);
      Delett(p^.A2);
         Dispose(p);
         p:=Nil;
  end;
 begin
   Delett(proot);
  Inherited  Destroy;
 end;

Procedure TtreeB.Write;
 Procedure Wr(p:Ptree); //Рекурсивный обход
    begin
   if p<>nil then
     begin
Form1.Memo1.Lines.Add(p^.Inf.s+
                          +IntToStr(p^.Inf.key));
          // Печать или запись в файл ставится здесь
    Wr(p^.A1);
    Wr(p^.A2);
     end;
  end;
 begin
 p:=proot;
  wr(p)
end;
End.


Unit Unit1;
. . . . . .
Var tr:TtreeB;

Procedure TForm1.FormCreate(Sender: TObject);
begin
n:=9; Edit1.Text:=IntToStr(n);
Memo1.Clear;
  With StringGrid1 do begin
Cells[0,0]:='Ф.И.О.';     Cells[1,0]:='Номер';
Cells[0,1]:='Иванов А.А.'; Cells[1,1]:='100005';
```

```
Cells[0,2]:='Петров С.И.'; Cells[1,2]:='100002';
Cells[0,3]:='Сидоров К.М.'; Cells[1,3]:='100004';
Cells[0,4]:='Васильев М.К.'; Cells[1,4]:='100001';
Cells[0,5]:='Смирнов В.К.';  Cells[1,5]:='100007';
Cells[0,6]:='Мишин Т.В.'; Cells[1,6]:='100006';
Cells[0,7]:='Долин А.К.'; Cells[1,7]:='100008';
Cells[0,8]:='Катаев А.М.'; Cells[1,8]:='100000';
Cells[0,9]:='Рубан В.В.'; Cells[1,9]:='100009';
                          end;
  tr:=TtreeB.create;
end;

Procedure TForm1.BitBtn1Click(Sender:TObject);
 begin          // Изменить количество элементов
  n:=StrToInt(Edit1.Text);
  StringGrid1.RowCount:=N+1;
end;

Procedure TForm1.BitBtn2Click(Sender: TObject);
 Var i:integer;        // Ввод данных в дерево
     a: TMs;           // и его отображение
 begin
    For i:=1 to n do
       With  A[i] do begin
           s:=StringGrid1.Cells[0,i];    // Ф.И.О.
           key:=StrToInt(StringGrid1.Cells[1,i]);
            end;
       tr.Make(a,n);
       tr.View;
 end;

Procedure TForm1.BitBtn3Click(Sender:TObject);
 begin           // Обход дерева
   Memo1.Clear;
   Tr.Write;
 end;

Procedure TForm1.BitBtn4Click(Sender:TObject);
 begin           // Удаление дерева
   tr.free;
   Memo1.Clear;     TreeView1.Items.Clear;
 end;
end.
```

8.5. Индивидуальные задания

Исходная информация в виде массива находится в компоненте StringGrid. Каждый элемент массива содержит строку текста и целочисленный ключ (например, Ф.И.О. и номер паспорта).

Разработать класс (`Unit2`) для работы с деревом поиска, содержащий следующие стандартные методы:

- 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.

На основе стандартного класса создать класс для решения задачи выбранного варианта.

Написать программу (`Unit1`), иллюстрирующую все методы работы с деревом поиска. Результат формирования и преобразования дерева показывать в компоненте `TTreeView`. Написать обработчик события, реализующий работу с методом решения своего варианта.

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, находящихся в узлах дерева.
8. Определить число листьев на каждом уровне дерева.
9. Определить число узлов в дереве, в которых есть указатель только на одну ветвь.
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 узел, имеющий ближайший к этому значению ключ.
13. Найти запись с ключом, ближайшим к среднему значению между максимальным и минимальным значениями ключей.
14. Определить количество узлов в левой ветви дерева.
15. Определить количество узлов в правой ветви дерева.

ТЕМА 9. ПРОГРАММИРОВАНИЕ С ИСПОЛЬЗОВАНИЕМ ХЕШИРОВАНИЯ

Цель лабораторной работы: получить навыки программирования алгоритмов с использованием хеширования.

9.1. Понятие хеширования

Имеется следующая проблема – поиск данных в списке:

а) если данные расположены беспорядочно в массиве (линейном списке, файле), то осуществляют **линейный поиск**, его эффективность $0(n/2)$;

б) если имеется упорядоченный массив (или 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.

Алгоритм хеширования реализуется следующим образом.

Допустим, имеется набор n записей с ключами в диапазоне $0 \le key \le K$, $n < K$. Выберем M немного большим, чем n.

Создадим массив (**хеш-таблицу**):

```
H: array[0..M-1] of <тип записей>;
```

Вначале его очистим **H[i]:=0** и наши n записей помещаем в этот массив в соответствии со значением ключа и функции $i=h(key)$, $0 \le i \le M-1$. Затем, используя операторы

```
zp:=H[h(key)]; // Извлекаем из массива
H[h(key)]:=zp; // Записываем в массив
```

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.

9.2. Порядок написания программы

Задание. Создать класс (в модуле `Uhesh`), реализующий стандартные методы работы с хеш-таблицей на основе массива стеков `Create, Free, Add, Read, Red, Print(TMemo)`.

9.2.1. Фрагмент программы

Листинг 9.1

```
Unit Uhesh;
Interface
 Uses Grids, SysUtils,  Dialogs;
Type
   Tkey=integer;
   Tinf=Record
     Fio:string;
     key:integer;
       end;
   Psel=^sel;
    sel=Record
    inf:Tinf;
      A:Psel;
        end;
   Ms=array[0..1] of Psel;
   Pms=^Ms;

   TH=class(Tobject)
     M,n:Word;
     sp,sp1:psel;
     H:Pms;
    Constructor create(M0:word);
    Destructor Free(Var stringgrid:TStringGrid);
    Procedure Add(Inf:Tinf);
    Procedure Read(key:Tkey; Var Inf:Tinf);
              …
   end;
Implementation

   Constructor TH.create(M0:word);
    Var i:word;
    begin   Inherited create;
       M:=M0; n:=0;
       Getmem(H,M*4);
       for i:=0 to M-1 do H[i]:=Nil;
    end;
```

```
Destructor TH.Free; // Прочесть и удалить
  Var i,j:word;      // все записи
 begin
   j:=0;
    for i:=0 to M-1 do
      While H[i]<>Nil do begin
            Inc(j);
          sp:=H[i];
    StringGrid.Cells[0,j]:=sp^.inf.Fio;
    StringGrid.Cells[1,j]:=IntToStr(sp^.inf.key);
          H[i]:= sp^.A;
          dispose(sp);
                              end;
   FreeMem(H,4*M);
end;

Procedure TH.Add;   // Добавить новую запись
  var i: integer;
      p:Psel;
begin
 i:=inf.key Mod M;
   New(p);  Inc(n);
    p^.Inf:=Inf;
    p^.A:=H[i];
  H[i]:=p;
end;

Procedure TH.Read;    // Чтение без удаления
  var i: integer;
 begin
   i:=key mod M;
    sp:=H[i];
 While (sp<>Nil) and
                 (sp^.inf.key<>key) do sp:=sp^.A;
  if sp<>Nil then inf:=sp^.inf
             else ShowMessage('ключ не найден');
end;
                 …
end.
```

9.3. Индивидуальные задания

Написать программу (`Unit1`), иллюстрирующую работу всех методов работы с хеш-таблицей (значение *M* выбирать по своему усмотрению). Результат

формирования и преобразования хеш-таблицы показывать в компоненте `TMemo` методом `Print(TMemo)` в виде строк, отображающих стеки. После этого на базе стандартного родительского класса написать свой класс (в `Unit2`), реализующий метод решения своего варианта. Написать обработчик события, реализующий вызов метода решения своего варианта.

1. Создать хеш-таблицу со случайными целыми ключами в диапазоне от –50 до +50 и преобразовать ее в две таблицы. Первая должна содержать только положительные ключи, а вторая – отрицательные.

2. Создать хеш-таблицу со случайными целыми ключами и удалить из него записи с четными ключами.

3. Создать хеш-таблицу со случайными целыми ключами в диапазоне от –10 до 10 и удалить из него записи с отрицательными ключами.

4. Создать хеш-таблицу со случайными целыми ключами и найти запись с минимальным ключом.

5. Создать хеш-таблицу со случайными целыми ключами и найти запись с максимальным ключом.

6. Подсчитать, сколько элементов хеш-таблицы со случайными ключами превышает среднее значение от всех ключей.

7. Создать хеш-таблицу из случайных целых чисел и найти в ней номер стека, содержащего минимальное значение ключа.

8. Создать хеш-таблицу из случайных целых чисел и найти в ней номер стека, содержащего максимальное значение ключа.

9. Создать хеш-таблицу со случайными ключами и распечатать все элементы в порядке возрастания ключа.

10. Создать хеш-таблицу со случайными ключами и распечатать все элементы в порядке убывания ключа

11. Создать хеш-таблицу со случайными ключами и определить, сколько элементов находится в каждом стеке.

12. Подсчитать, сколько элементов хеш-таблицы со случайными ключами не превышает среднее значение от всех ключей.

13. Создать хеш-таблицу из случайных целых чисел и из нее сделать еще две. В первую поместить записи с ключами большими K, а во второй – с меньшими K.

14. Создать хеш-таблицу со случайными ключами в диапазоне от 1 до 10 и определить наиболее часто повторяющийся ключ.

15. Создать хеш-таблицу со случайными ключами и удалить из нее записи с ключами из диапазона $K1 < key < K2$.

ТЕМА 10. РАБОТА С РАЗРЕЖЕННЫМИ МАТРИЦАМИ

Цель лабораторной работы: получить навыки организации хранения и программирования алгоритмов обработки операций над разреженными матрицами с использованием массива из стеков.

10.1. Применение разреженных матриц

Решение многих прикладных задач (методы теории графов, численные методы решения дифференциальных уравнений методом конечных элементов, методы обработки изображений, криптографии) связано с обработкой матриц большой размерности n имеющих малое число ненулевых элементов. Матрицы, содержащие большое число нулевых элементов называются *разреженными*. Использование специальных способов размещения в памяти разреженных матриц и разработка специальных алгоритмов работы с ними позволяет во многих случаях существенно снизить затраты компьютерных ресурсов и уменьшить время вычислений. Одним из рациональных способов хранения ненулевых элементов матрицы является массив из динамических списков. Эффективность работы с таким массивом определяется введением класса с продуманным набором методов и создание на этой основе эффективных процедур работы с матрицами.

10.2. Порядок написания программы

Задание. Создать класс, реализующий стандартные методы записи элемента матрицы a_{ij} в стековую динамическую структуру, чтение элемента a_{ij}, отображения матрицы в компоненте `TStringGrid.` После этого дополнить класс необходимыми методами и создать процедуру, реализующую указанную в индивидуальном задании операцию с матрицами.

10.2.1. Пример оформления класса со стандартным минимальным набором методов

Листинг 10.1

```
unit URazMat;
Interface
 uses Grids,SysUtils;
Type
 TInf=Record     // Информационная ячейка содержит
 aj:extended;    // ненулевой  элемент и номер столбца
 j:word;
      end;
```

```
TSel=^sel;    // Ячейка с косвенной адресацией
  Sel=Record
  inf:tinf;
  A:TSel;
      end;

TLists=class(TObject) // Динамический список
  sp1,sp:TSel;   // Указатели: на начало и рабочий
  Procedure adds(inf:Tinf); // Добавить в начало стека
  Procedure addsj(inf:tinf); // Добавить в
                                  // отсорированный по индексу j стек
  Function readsj(j:word):extended; // Читать
  Procedure dels; //  Удаление первого элемента стека
       end;

Tmslists= array[1..1] of TLists;

TListms=class(TLists)     // Массив из списков
   ms:^Tmslists; // Указатель на динамический массив
   m,n:word;     // Размерность матрицы m x n
   constructor create(m0,n0:word);
   destructor free;
   Procedure add(i,j:word;a:extended);
   Procedure addj(i,j:word;a:extended);
   Function read(i,j:word):extended;
   Procedure Print(StG:TStringGrid);
        end;

 Tbx=array[1..1] of extended; // Решение СЛАУ с верхней
 Tbxp=^Tbx;                   // треугольной матрицей
 TSlau=class(TListms)
    b,x:Tbxp;
   constructor create(n0:word);
   Procedure Addb(i:word;bi:extended);
   Procedure SlauT;
   Procedure Print(Stg1,Stg2,Stg3:TStringGrid);
   destructor free;
         end;

Implementation

Function Tlists.readsj;
begin
```

```
 sp:=sp1;
  while (sp<>Nil) and (sp^.inf.j<>j) do sp:=sp.A;
 if sp=nil then result:=0
           else result:=sp^.Inf.aj;
 end;

Procedure TLists.Adds;
 begin
   New(sp);
   sp^.Inf:=Inf;    sp^.A:=sp1;    sp1:=sp;
end;

Procedure TLists.addsj;
 var spt:Tsel;j:word;
begin
  New(sp);
  sp^.Inf:=inf;
  if sp1=nil then begin sp1:=sp; sp1^.A:=nil end
             else
   if sp1^.inf.j>=inf.j then
              begin  sp^.A:=sp1; sp1:=sp end
                        else begin
             spt:=sp1;
          while(spt^.A<>nil) and (spt^.A^.inf.j<inf.j)
                                       do spt:=spt^.A;
            sp^.A:=spt^.A; spt^.A:=sp;
          end;
end;

Procedure TLists.dels;
 begin
sp:=sp1;  sp1:=sp1^.A;
Dispose(sp);
 end;

constructor TListms.create;
 Var i:word;
 begin m:=m0; n:=n0;
inherited create;
  GetMem(ms,4*m);
  for i:=1 to m do begin
     ms[i]:=Tlists.create; ms[i].sp1:=nil end;
 end;
```

```
constructor TSlau.create;
 begin
      inherited create(n0,n0);
    GetMem(b,sizeof(extended)*n0);
    GetMem(x,sizeof(extended)*n0);
 end;

destructor TListms.free;
 Var i:word;
 begin
   for i:=1 to m do
    while ms[i].sp1<>nil do ms[i].dels;
     FreeMem(ms,4*m);
 end;

destructor TSlau.free;
 begin
    FreeMem(x,sizeof(extended)*n);
    FreeMem(b,sizeof(extended)*n);
    inherited free;
 end;

Procedure TListms.Print;
 Var i,j:word;
 begin
for i:=1 to m do
for j:=1 to n do
    StG.Cells[j,i]:=FloatToStr(ms[i].readsj(j))
 end;

Procedure TListms.add;
 Var inf:Tinf;
 begin
   inf.aj:=a; inf.j:=j;
   ms[i].adds(inf);
 end;

Procedure TListms.addj;
 var inf:Tinf;
 begin
   inf.aj:=a; inf.j:=j;
   ms[i].addsj(inf);
 end;
```

```
Function TListms.read;
 Var inf:Tinf;
 begin
   result:=ms[i].readsj(j);
 end;

Procedure TSlau.Addb;
begin
b[i]:=bi;
end;

Procedure TSlau.Print;
 Var i,j:word;
 begin
  inherited Print(stg1);
for i:=1 to m do begin
   StG2.Cells[0,i]:=FloatToStr(b[i]);
   StG3.Cells[0,i]:=FloatToStr(x[i]);
                 end;
 end;

Procedure TSlau.SlauT;
  var a1,s:extended;pt,sp:Tsel;i,j:word;
  begin
   pt:=ms[n].sp1;
   if pt<>nil then a1:=pt^.inf.aj else a1:=0;
   x[n]:=b[n]/a1;
  for i:=n-1 downto 1 do begin
   sp:=ms[i].sp1;  a1:=sp^.inf.aj;  s:=0;  sp:=sp^.A;
     while sp<>nil do begin
      j:=sp^.inf.j;  s:=s+sp^.inf.aj*x[j];  sp:=sp^.A;
                    end;
       x[i]:=(b[i]-s)/a1;
                          end;
 end;
end. // Конец UrazMat

unit matr;
          …
uses   …,  URazMat;

procedure TForm1.BitBtn1Click(Sender: TObject);
 Var Var G:Tslau;
```

```
Begin
G:=TSlau.create(2);
G.addj(1,1,2);  G.addj(1,2,4);  G.addj(2,2,3);
G.Addb(1,6);    G.Addb(2,3);
G.SlauT;
G.print(StringGrid1,StringGrid2,StringGrid3);
G.free;
End;
end.
```

10.3. Индивидуальные задания

Решить поставленную задачу и продемонстрировать возможности разработанного программного продукта. Доказать, что разработан алгоритм, использующий минимально возможное число проверок и операций.

1. Создать метод чтения и удаления диагональных элементов матрицы.
2. Создать процедуру получения транспонированной матрицы, при этом исходная матрица должна сохраняться.
3. Создать процедуру умножения двух матриц. Исходные матрицы сохранить. Полезно использовать отсортированность стека по индексу j.
4. Создать метод перестановки двух заданных строк в матрице.
5. Создать метод перестановки двух заданных столбцов в матрице с упорядоченными по возрастанию индекса j элементами в стеке.
6. Создать метод нахождения максимального элемента матрицы.
7. Создать процедуру получения одномерной разреженной матрицы, содержащей максимальные элементы по каждому столбцу исходной матрицы.
8. Создать процедуру получения одномерной разреженной матрицы (вектора-столбца), содержащей максимальные элементы по каждой строке исходной матрицы.
9. Создать метод нахождения индексов максимального элемента матрицы.
10. Создать метод `readd(i,j)` чтения и удаления из матрицы элемента a_{ij}.
11. Создать метод нахождения суммы диагональных элементов матрицы.
12. Создать процедуру, реализующую прямой ход метода Гаусса, приводящий исходную матрицу к верхней треугольной матрице, и получить определитель.
13. Создать метод нахождения количества ненулевых элементов в матрице и их среднего арифметического значения.
14. Имеется система уравнений $A\vec{x} = \vec{b}$, причем заданы A – верхняя треугольная разреженная матрица и вектор-столбец $\vec{b}$. Составить процедуру нахождения решения в виде вектора-столбца $\vec{x}$ (см. обратный ход метода Гаусса).
15. Создать процедуру получения из одной матрицы двух. Одна содержит элементы исходной матрицы, меньшие среднеарифметического значения, другая состоит из элементов исходной матрицы, больших среднеарифметического значения. Нулевые элементы не проверять.

ЛИТЕРАТУРА

1. Синицын, А. К. Основы алгоритмизации и программирования в среде DELPHI. Алгоритмы на структурах данных : лаб. практикум по курсу «Основы алгоритмизации и программирования» для студ. 1–2-го курсов всех спец. БГУИР / А. К. Синицын, А. А. Навроцкий. – Минск : БГУИР, 2005.

2. Синицын, А. К. Конспект лекций по курсу «Программирование» для студ. 1–2-го курсов всех спец. БГУИР / А. К. Синицын. – Минск : БГУИР, 2001.

3. Вирт, Н. Алгоритмы и структуры данных / Н. Вирт – СПб. : «Невский диалект», 2005.

4. Стивенс, Р. Delphi. Готовые алгоритмы / Р. Стивенс. – М. : ДМК Пресс; СПб. : Питер, 2004.

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

6. Бакнелл, Дж. Фундаментальные алгоритмы и структуры данных в Delphi / Дж. Бакнелл. – СПб. : Питер, 2006.

7. Кнут, Д. Искусство программирования. В 3 т. Т 3. Сортировка и поиск / Д. Кнут – М. : Вильямс, 2000.

8. Хопкрофт, Дж. Структуры данных и алгоритмы / Дж. Хопкрофт, Дж. Ульман, А. Ахо – М. : Вильямс, 2003.

9. Синицын, А. К. Основы алгоритмизации и программирования в среде DELPHI. Электронный учебно-методический комплекс / А. К. Синицын, А. А. Навроцкий. – Минск : БГУИР, 2006.

Св. план 2007, поз. 54

Учебное издание

Синицын Анатолий Константинович
Навроцкий Анатолий Александрович

ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ В СРЕДЕ DELPHI. АЛГОРИТМЫ НА СТРУКТУРАХ ДАННЫХ

Лабораторный практикум по курсу
«Основы алгоритмизации и программирования»
для студентов 1–2-го курсов всех специальностей БГУИР

Редактор Е. Н. Батурчик
Корректор М. В. Тезина

Подписано в печать 29.08.2007. Формат 60x84 1/16. Бумага офсетная.
Гарнитура «Таймс». Печать ризографическая. Усл. печ. л. 3,14.
Уч.-изд. л. 2,7. Тираж 500 экз. Заказ 161.

Издатель и полиграфическое исполнение: Учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники»
ЛИ №02330/0056964 от 01.04.2004. ЛП №02330/0131666 от 30.04.2004.
220013, Минск, П. Бровки, 6