ФЕДЕРАЛЬНОЕ АГЕНТСТВО ВОЗДУШНОГО ТРАНСПОРТА
(РОСАВИАЦИЯ)

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ ГРАЖДАНСКОЙ АВИАЦИИ» (МГТУ ГА)

Кафедра прикладной математики

А.А. Егорова

АЛГОРИТМИЧЕСКИЕ ЯЗЫКИ И ПРОГРАММИРОВАНИЕ

ЧАСТЬ III. ДИНАМИЧЕСКАЯ ПАМЯТЬ

Учебно-методическое пособие
по выполнению лабораторных работ

для студентов II курса
направления 01.03.04
очной формы обучения

Москва
ИД Академии Жуковского
2021

УДК 004.42
ББК 6Ф7.3
Е30

Рецензент:
Романчева Н.И. – канд. техн. наук, доцент

Егорова А.А.
Е30 Алгоритмические языки и программирование. Часть III. Динамическая память [Текст] : учебно-методическое пособие по выполнению лабораторных работ / А.А. Егорова. – М.: ИД Академии Жуковского, 2021. – 52 с.

Данное учебно-методическое пособие издается в соответствии с рабочей программой учебной дисциплины «Алгоритмические языки и программирование» по учебному плану для студентов II курса направления 01.03.04 очной формы обучения.

Рассмотрено и одобрено на заседаниях кафедры 19.10.2021 г. и методического совета 19.10.2021 г.

УДК 004.42
ББК 6Ф7.3

В авторской редакции

Подписано в печать 08.12.2021 г.
Формат 60х84/16 Печ. л. 3,25 Усл. печ. л. 3,02
Заказ № 874/1004-УМП33 Тираж 25 экз.

Московский государственный технический университет ГА
125993, Москва, Кронштадтский бульвар, д. 20

Издательский дом Академии имени Н. Е. Жуковского
125167, Москва, 8-го Марта 4-я ул., д. 6А
Тел.: (495) 973-45-68
E-mail: zakaz@itsbook.ru

ПРЕДИСЛОВИЕ

Настоящее пособие содержит задания на выполнение лабораторных работ по дисциплине «Алгоритмические языки и программирование», выполняемых студентами 2 курса направления подготовки «Прикладная математика» во втором семестре.

Навыки, приобретенные в процессе выполнения лабораторных работ, необходимы студентам в процессе дальнейшей подготовки практически по всем дисциплинам, а также для самостоятельной работы и самоподготовки.

В пособии отражены организационно-методические аспекты выполнения лабораторных работ, особенности их проведения, цели, достигаемые в процессе выполнения лабораторных работ.

Пособие охватывает дисциплину не полностью, а только третьего семестра и содержит цель, задание на выполнение каждой лабораторной работы, требования к их выполнению, краткие теоретические сведения, примеры выполнения и контрольные вопросы.

Теоретический материал содержит информацию по работе с динамической памятью, разработке линейных динамических структур, приемам работы с графами, основам объектно-ориентированного программирования. Теоретический материал не претендует на полноту, а содержит только необходимую для выполнения работы информацию. Требования к выполнению работ являются обязательными в рамках выполнения работ по дисциплине «Алгоритмические языки и программирование», сформулированы в соответствии с методикой преподавания дисциплины.

Список рекомендуемой литературы содержит актуальные ссылки на Интернет-источники.

Настоящее пособие может быть использовано и как справочник при самостоятельной работе при программировании прикладных задач, в том числе по другим дисциплинам.

Пособие имеет прикладной характер, что способствует формированию у студентов компетенций в соответствии с требованиями, содержащимися в рабочей программе по дисциплине «Алгоритмические языки и программирование», и в целом соответствующие модели компетенций по направлению подготовки «Прикладная математика».

Оглавление

1. Введение

При программировании навык решения задач, способность воспринять описание поставленной задачи, написать программный код для ее решения – один из основных навыков, который должен «наработать» будущий IT-специалист.

Настоящее пособие, предназначенное для выполнения лабораторных работ по дисциплине «Алгоритмические языки и программирование», призвано, в первую очередь, привить студентам навыки грамотного построения программ с использованием инструментов для минимизации объема занимаемой памяти, сокращения времени обработки, повышения читабельности программ.

Целью проведения лабораторных работ является как закрепление основных теоретических положений, изложенных в лекциях, так и получение практических навыков по реализации грамотного использования динамической памяти, построения линейных динамических структур, обработки графов и построения и использования объектов.

Несмотря на то, что программирование – процесс творческий, он имеет в своей основе еще две составляющие: аналитическую и синтаксическую. Темы, рассматриваемые в настоящем пособии, обеспечивают возможность развития всех трех составляющих. Глобальная цель дисциплины «Алгоритмические языки и программирование» - научиться системному подходу к решению задач по программированию и быть уверенным в том, что вы сможете подойти к решению задачи оптимальным образом, выбрать соответствующие задаче инструменты, а в итоге решить любую задачу.

Большое значение имеет стиль написания программ. И несмотря на то, что автор не подчеркивает этот вопрос в пособии, в процессе выполнения лабораторной работы следует уделять ему большое внимание, обращая внимание на примеры написания программ в пособии, выбор инструментов, их взаимосвязь и правила использования.

Примеры в пособии написаны на языке PascalABC.net, но все рассмотренные темы могут быть реализованы на любом языке программирования высокого уровня, так как практически все теоретические вопросы являются базовыми и их реализации различаются синтаксисом.

Пособие необходимо студентам на всех этапах, начиная от подготовки до оформления отчета и защиты лабораторной работы.

2. Организационно-методические рекомендации

2.1. Отчет по лабораторной работе

По окончании лабораторной работы студент оформляет отчет, который включает:

1. номер работы,
2. название работы,
3. цель работы,
4. задание на выполнение работы,
5. вариант выполнения работы,
6. алгоритмы выполнения каждой процедуры за исключением небольших линейных (допускается «от руки»),
7. программа (листинг, распечатка),
8. распечатки исходных данных и результатов работы (или файлы, или скриншоты в соответствии с заданием).

Если лабораторная работа предполагает выполнение нескольких заданий, то п.5-8 группируются по каждому заданию.

Скриншоты делать не полного экрана, а фрагмента, захватив актуальную информацию. Распечатка программы должна содержать комментарии: фамилию студента и другие, необходимые для удобочитаемости программы. Размеры букв в распечатке должны

быть такими, чтобы текст было удобно читать, не поднимая лист со стола (например, не менее 10 пт для шрифта Times New Roman). Схема алгоритма, выполненная «от руки» должна быть аккуратной, использование линейки обязательно.

Номер работы и ее название, а также фамилия и группа студента оформляются на титульном листе, где также указывается дисциплина.

2.2. Защита лабораторной работы

После сдачи выполненной работы преподавателю и оформления отчета студент защищает работу (на следующем занятии). В процессе защиты преподаватель проверяет правильность оформления отчета (в первую очередь алгоритмов), соответствие программы заданию, исходных данных и результатов программе. Студент отвечает на вопросы преподавателя как по выполнению работы, так и по теоретической части работы (цель работы и контрольные вопросы - ориентир).

3. Лабораторная работа №10

Разработка программы для работы с динамическими переменными

3.1. Цель работы

Целью работы является освоение:

- объявления указателей на массивы арифметических данных и массивы записей;
- обращения к значениям одномерного и многомерного массива;
- процедуры выделения ОП;
- процедуры освобождения ОП;
- формирование двумерного массива с помощью указателя на одномерный массив и на двумерный массив;
- формирования динамического массива с помощью процедуры и функции.

3.2. Задание на выполнение работы

Лабораторную работу выполнить в соответствии со своим вариантом на базе лабораторных работ №1, № 3 и №4 второго семестра: обработка двумерного арифметического массива, включая сортировку, и обработка массива записей, включая сортировку:

1. обработка массивов арифметических данных с использованием процедур с выделением и освобождением ОП из кучи;
2. обработка данных массивов записей, с выделением и освобождением ОП для массива записей;
3. выполнить программу для 3-4 комплектов исходных данных: нечетные варианты – задание 1, четные варианты – задание 2. Сформировать общий массив из результатов обработки массива или поиска (извлечений) по каждому комплекту в соответствии с вариантом.

3.3. Требования к выполнению

- Поиск использовать любой;
- Двумерный массив объявлять как одномерный;
- Формируемые массивы должны быть динамическими;

– В отчете представляются схемы алгоритмы процедур и основной программы в виде графических схем только в том случае, если он и отличаются от схем, разработанных ранее при выполнении лабораторных работ.

3.4. Краткие теоретические сведения

Память для переменных может быть выделена статически, т.е. на этапе компиляции, и динамически, в процессе выполнения программы. В зависимости от этого и переменные называются статическими и динамическими.

За статическими переменными память закрепляется на все время выполнения программы, даже если они больше не используются. Если объявлено несколько больших массивов, памяти может не хватить. Выход из положения – использование динамической памяти.

Динамическая память (куча) необходима в следующих случаях:

1. Для больших массивов, которые используются одновременно.
2. Для массивов, требуемый объем памяти, для которых определяется в процессе выполнения программы.
3. Для временного запоминания графических или звуковых данных.

Границы статического массива изменить в ходе программы нельзя. Основным недостатком такого массива является то, что в случае, когда заранее не известно количество элементов массива, то приходится выделять память максимального размера, иногда гораздо больше, чем требуется.

Рассмотрим работу с динамическим массивом.

Объявление динамического массива:

```
var a: array of integer;
```

Массив можно объявить и в теле программы:

```
begin
...
var a: array of integer;
```

Или объявление с инициализацией:

```
var a:= Arr(9,8,7,6);
```

Выделение памяти и размер такого массива задается уже по ходу программы одним из трех способов:

```
var a: array of integer;
var n:=readInteger;
a:=new integer[n];
```

```
var a: array of integer;
a:=new integer[readInteger];
```

```
var a: array of integer;
var n:=readInteger;
SetLength(a,n); // устанавливаем размер массива a
```

Ссылочная объектная модель: память выделяется служебным словом NEW

```
var a := new integer[5];
```

Возможна инициализация динамического массива при описании:

```
var a: array of integer := (9,8,7);
```

Новые способы заполнения массива:

```
var a:=Arr(9,8,7);// по правой части - integer
var a:=ArrFill(5,2); // 2 2 2 2 2
```

Ввод с клавиатуры:

```
var a:=ReadArrInteger(5);
var a:=ReadArrReal(5);
```

Заполнение случайными числами:

```
var a:=new integer[10];
a:=arrRandomInteger(10);
```

Вывод массива: `print(a);`
Или с разделителем между элементами: `print(a, '; ');`
Или: `printArr(a);`

Допускается переприсваивание массивов:

```
var a: array of integer := (1,2,3);

var b:=a;
```

Если теперь изменить значение элементов массива b, то изменится и массив a. Для того чтобы избежать данной ситуации, необходимо создать массив b, как копию массива a:

```
var a: array of integer := (1,2,3);

var b:=Copy(a);

b[2]:=1;

print(a); //[1,2,3]
```

Очистка динамического массива: `a:=nil;`
В процессе очистки выполнение программы и всех процессов приостанавливается.

Работа с элементами массива:

1. Цикл for

```
for var i:=0 to High(a) do
   print(a[i]);;
```

2. Цикл foreach

```
foreach var x in a do
    print(x);
```

`High(массив)` — возвращает верхнюю границу динамического массива.

Динамические массивы позволяют управлять количеством элементом в каждом из их измерений во время выполнения программы.

Пример объявления массива:	**Статический**	**Динамический**	**Вызов SetLength (для динамического массива)**
Одномерный	`<имя>: array [0..N - 1] of <тип>`	`<имя>: array of <тип>`	`SetLength(<имя>, N)`
Двумерный	`<имя>: array [0..N -`	`<имя>: array [,]`	`SetLength(<имя>, N, M)`

	1, 0..M - 1] of <тип>	of <тип>	
Трехмерный	<имя>: array [0..N - 1, 0..M - 1, 0..K - 1] of <тип>	<имя>: array [,,] of <тип>	SetLength(<имя>, N, M, K)

3.5. Пример выполнения работы

Рассмотрим выполнение лабораторной работы на следующем примере.

Дан двумерный массив X. Требуется ввести и вывести исходный массив, а также:

1) Сформировать массив из нечетных строк массива;
2) Сформировать массив из четных строк массива;
3) Сформировать массив из нечетных столбцов массива;
4) Сформировать массив из четных столбцов массива;
5) Сформировать массив из сумм элементов четных столбцов;

В приведенной ниже программе для ввода массива приведены и функция, и процедура для демонстрации возможностей программы. Для каждой обработки, а также для вывода массива использована своя процедура.

```
PROGRAM Dinpam_1;

 TYPE    AP = Array of REAL;
 VAR     A, AN : STRING[80];
         B, C , D, R: AP;
         M, N, I, J : INTEGER;
         MIN, MAX : REAL;
         FI : TEXT;

{ --------Процедура вывода--------------------}

     PROCEDURE P(B : AP; M, N : INTEGER);
        var i,j : integer;
     BEGIN
          For I := 0 to M-1 DO                BEGIN
              FOR J := 0 to N-1 DO WRITE ( B[I * N + J]: 12:4 );
              WRITELN;  end;
            End;

{ --------Процедура ввода--------------------}
    PROCEDURE VVOD ( VAR B: AP; VAR M, N : INTEGER );
      var i : integer;
    BEGIN
        READLN (FI, M, N ); WRITELN ( 'M = ', M:3, '  N= ', N:3);
        SetLength(B, M*N);
         FOR I := 0 TO M*N-1 DO READ ( FI, B[i]);
    END;

{ --------Функция ввода--------------------}
function VVOD1 ( VAR M, N : INTEGER ): AP;
      var i : integer; b: AP;
    BEGIN
        READLN (FI, M, N ); WRITELN ( 'M = ', M:3, '  N= ', N:3);
        SetLength(B, M*N);
         FOR I := 0 TO M*N-1 DO READ ( FI, B[i]);
     VVOD1:=B;
   END;

   { --------Процедура формирования массива из нечетных строк--------------}
    PROCEDURE FORMSTROK ( VAR B,C: AP; VAR M, N : INTEGER );
      var i, j, k, M1 : integer;
    BEGIN
        M1:= (M+1) div 2; //WRITELN (M1);
        k:=0;
```

```
        SetLength(C, M1*N);
         For I := 0 to M1-1 DO                  BEGIN
              FOR J := 0 to N-1 DO C [I * N + J] := B[k * N + J];
               k:=k+2;
               end;
     END;

  { --------Процедура формирования массива из четных строк--------------------}
    PROCEDURE FORMSTROK2 ( VAR B,C: AP; VAR M, N : INTEGER );
      var i, j, k, M1, k1, k2 : integer;
    BEGIN
        M1:= (M+1) div 2; //WRITELN (M1);
        k:=1;
        SetLength(C, M1*N);
         For I := 1 to M1 DO                  BEGIN
              FOR J := 0 to N-1 DO begin
                C [(I-1) * N + J] := B[k * N + J];
         //       writeln ((I-1) * N + J :4, k * N + J:4);
               end;
              k:=k+2;                 end;
     END;

{ --------Процедура формирования массива из четных столбцов ----------------}
    PROCEDURE  FORMSTOLB ( VAR B,D: AP; VAR M, N : INTEGER );
      var i, j, k1, k2, N1 : integer;
    BEGIN
        N1:= N div 2; WRITELN (N1);
        SetLength(D, N1*M);
        k1:=1;
        FOR j := 0 to N1-1 DO BEGIN
           k2:=k1;
           For i := 0 to M-1 DO BEGIN
               WRITELN ( [i * N1 + j] ,  B[k2]: 12:4 );
               D [i * N1 + j] :=  B[k2];
                k2:=k2+N;
                                    end;
           k1:=k1+2;
                               end;
    END;

{ --------Процедура формирования массива из нечетных столбцов --------------}
     PROCEDURE  FORMSTOLB2 ( VAR B,D: AP; VAR M, N : INTEGER );
          var i, j, k1, k2, N1 : integer;
     BEGIN
        N1:= (N + 1) div 2; WRITELN (N1);
        SetLength(D, N1*M);
        k1:=0;
        FOR j := 0 to N1-1 DO BEGIN
           k2:=k1;
           For i := 0 to M-1 DO BEGIN
               WRITELN ( [i * N1 + j] ,  B[k2]: 12:4 );
               D [i * N1 + j] :=  B[k2];
                k2:=k2+N;
                                    end;
           k1:=k1+2;
                               end;
    END;
{ --Процедура формирования массива из сумм элементов четных столбцов--------}
    PROCEDURE  FORMSUMM ( VAR B,D: AP; VAR M, N : INTEGER );
      var i, j, k1, k2, M1 : integer;
    BEGIN
        M1:= (N+1) div 2; //WRITELN (M1);
        k1:=1;
        SetLength(D, M1*M);
        For J := 0 to M1-1 DO BEGIN
              D[J] := 0; k2:=1;
```

```
                FOR I := 0 to M-1 DO BEGIN
                 // WRITELN (B[k1+k2-1]);
                   D[J] := D[J] + B[k1+k2-1];
                   k2:=k2+N;
                                        end;
                k1:=k1+2;
                                   end;
    END;

 {-------------- Основная программа--------------}
BEGIN   ASSIGN (FI, 'LR5.DAT');  RESET(FI);
  FOR I := 1 TO 4 DO  BEGIN
           READLN (FI, A );  WRITELN ( A );
                       END;  READLN  (FI, A );
                             READLN  (FI, AN);
  //VVOD (B, M, N );
  B:= VVOD1 (M, N );
  P ( B, M, N );
  WRITELN  ( AN);
  WRITELN ('         Сформированные массивы');
   FORMSTROK (B,C, M, N); WRITELN;
   P ( C, (M+1) div 2, N );
   FORMSTROK2 (B,C, M, N); WRITELN;
   P ( C, (M+1) div 2, N );
   FORMSTOLB (B,D, M, N); WRITELN;
   P ( D, M, (N) div 2 );
   FORMSTOLB2 (B,D, M, N); WRITELN;
   P ( D, M, (N+1) div 2 );
   FORMSUMM (B,R, M, N); WRITELN;
   P ( R, 1, (N+1) div 2 );

   B:=nil; C := nil; D:= nil; R:= nil;
   FREEMEM ( B, M * N * SIZEOF ( REAL ) );
END.
```

Исходные данные представлены на листинге:

Stek.pas | 1.txt | r.txt | •Perevod.pas | steck1.txt | steck2.txt | ProgDP-8.pas | LR5.DAT

```
                Двумерный массив
================================================================================
!     1      !      2     !     3      !      4     !      5     !      6     !
================================================================================
--------------------------------------------------------------------------------
================================================================================
4 5
 1.4   23.9   312        47.4       563.2       4E4
 .007  67.33   9       -2.446      -1.65       1.5555555E-4
 4.2    95.3  6.44   -7.52345   5.23456      1.2E-2
1.007  67.33   9.33      -2.446      -1.6       1.5555555E-4
 4.2    95.3  6.44   -7.52345   5.23456      1.2E-2

```

Результат работы программы представлен на листинге:

```
Окно вывода
                Двумерный массив
=============================================================================
!     1      !      2     !      3     !      4     !      5     !      6     !
=============================================================================
M =    4  N=    5
      1.4000      23.9000     312.0000      47.4000     563.2000
  40000.0000       0.0070      67.3300       9.0000      -2.4460
     -1.6500       0.0002       4.2000      95.3000       6.4400
     -7.5235       5.2346       0.0120       1.0070      67.3300
=============================================================================
        Сформированные массивы

      1.4000      23.9000     312.0000      47.4000     563.2000
     -1.6500       0.0002       4.2000      95.3000       6.4400

  40000.0000       0.0070      67.3300       9.0000      -2.4460
     -7.5235       5.2346       0.0120       1.0070      67.3300
2
{0}      23.9000
{2}       0.0070
{4}       0.0002
{6}       5.2346
{1}      47.4000
{3}       9.0000
{5}      95.3000
{7}       1.0070

     23.9000      47.4000
      0.0070       9.0000
      0.0002      95.3000
      5.2346       1.0070
3
{0}       1.4000
{3}   40000.0000
{6}      -1.6500
{9}      -7.5235
{1}     312.0000
{4}      67.3300
{7}       4.2000
{10}       0.0120
{2}     563.2000
{5}      -2.4460
{8}       6.4400
{11}      67.3300

      1.4000     312.0000     563.2000
  40000.0000      67.3300      -2.4460
     -1.6500       4.2000       6.4400
     -7.5235       0.0120      67.3300
```

Здесь цифры в фигурных скобках означают порядковый номер в одномерном массиве.

3.6. Варианты на выполнение работы №10

Задание №1

Выделение и освобождение ОП одномерному массиву для работы с ним как с двумерным, а также ввод массива осуществлять из:

а) главной программы;
б) процедуры;
в) функции

в соответствии со своим вариантом:

а	1, 4, 7, 10, 13, 16, 19
б	2, 5, 8, 11, 14, 17, 20, 22
в	3, 6, 9, 12, 15, 18, 21

Задание №2

Выделение и освобождение ОП массиву записей, а также ввод массива осуществлять из:
а) из главной программы;
б) из процедуры;
в) из функции

в соответствии со своим вариантом:

а	2, 5, 8, 11, 14, 17, 20
б	3, 6, 9, 12, 15, 18, 21
в	1, 4, 7, 10, 13, 16, 19, 22

3.7. Контрольные вопросы

1. Что такое указатель?
2. Что такое динамическая переменная, как ее объявить?
3. Объясните механизм выделения памяти под динамическую переменную.
4. Поясните назначение и использование функции NEW.
5. Как объявить динамический массив?
6. Что такое куча?
7. Как освободить память от динамической переменной?
8. Как возвратить из функции/процедуры указатель на динамический массив?
9. Поясните способ работы с многомерным массивом, объявив его одномерным.
10. Поясните назначение функции SetLength.
11. Когда необходимо использовать динамические переменные, когда их использовать желательно?

4. Лабораторная работа №11

Разработка программ реализации динамических связных линейных структур данных

4.1. Цель работы

Целью выполнения данной лабораторной работы является освоение приемов:

— разработки алгоритмов использования структуры стека, очереди и списка

— реализация их на языке высокого уровня (Паскаль) с использованием динамического выделения памяти для реализации структуры типа «стек», «очередь», «список».

4.2. Задание на выполнение работы

1. Разработать алгоритм и программу в соответствии со своим вариантом, реализующую заданную линейную структуру динамического типа (стек и/или очередь или список).
2. Разработать алгоритм и программу, выполняющую создание списка с элементами заданной структуры (в соответствии со своим вариантом из ЛР №4 второго семестра), а также выполняющую следующие процедуры:
   - поиск по уникальному и неуникальному признаку,
   - удаление из списка,
   - чтение элементов списка в двух направлениях.

4.3. Требования к выполнению работы

1. Программы должны быть хорошо структурированы, и содержать процедуры в соответствии со своим вариантом:
   - проверки пустоты стека (очереди/ списка);
   - добавления элемента в стек (очередь/ список);
   - удаления элемента из стека (очереди/ списка);
   - чтения элемента стека (очереди/ списка).
2. В программах необходимо использовать динамическое выделение памяти для связывания элементов.
3. Вся обработка в соответствии с вариантом должна применяться к динамической структуре после ее создания, ни в коем случае не в процессе создания.
4. Все списки упорядочены.

4.4. Краткие теоретические сведения

Под структурой данных будем понимать способ организации и хранения данных, определяющий доступ к этим данным и их модификацию.

К основным структурам данных (или структурам для данных) обычно относят массивы, записи и множества. Основные они потому, что из них можно образовывать более сложные структуры, и они чаще встречаются на практике.

Записи, содержащие указатели, позволяют формировать линейные и нелинейные связанные структуры.

При объявлении данных фиксируется диапазон значений, присваиваемых этим переменным, размер выделяемой памяти и определяются допустимые операции. О таких переменных говорят, как о статических переменных.

Однако целый ряд задач предъявляет к данным иные требования. Не всегда на момент начала решения возможно определить, например, размер выделяемой памяти, или в процессе выполнения требуется ее изменить. И не только объем памяти, а и саму структуру. О таких структурах говорят как о динамических.

Таким образом, можно определить данные с динамической структурой как переменные сложного типа (составные), структура которых меняется непосредственно в процессе выполнения программы. В зависимости от конструкции и допустимых операций динамические структуры данных можно разделить на 2 группы: линейные и нелинейные.

К линейным связным структурам относятся стеки, очереди и списки. Линейные структуры они представляют собой последовательность элементов, связанных между собой в цепь с помощью указателя; число элементов структуры ограничено только ОП; память под каждый элемент выделяется динамически; структура имеет начало, адресуемое указателем.

Все линейные структуры имеют аналогичную структуру. Стек, очередь и список можно объявить, используя запись, состоящую как минимум из двух полей: информационного и адресного. При этом адресное поле указывает на элемент такой же структуры, как и объявляемый. Структуры данных такого типа называют еще рекурсивными. Информационное же поле (поле данных) может быть какого угодно сложного типа, кроме процедурного. А можно включать в состав такой записи целый ряд полей данных.

Объявление линейной структуры:

```
Type
PSTRC = ^ STRC;
STRC = record
INF : integer;     // <- информация элемента
PTR : PSTRC     // <- указатель на предыдущий элемент
      end;

Var S: PSTRC;
```

Исключение составляют двунаправленные списки, имеющие в своем составе два адресных поля.

Таким образом, можно выделить общие свойства линейных динамических структур:

— все элементы таких структур представляют собой последовательность элементов, связанных между собой указателями;

— количество элементов ограничивает только объем оперативной памяти;

— оперативная память для элементов выделяется динамически;

— последовательность элементов всегда имеет начало и может иметь конец, которые обычно адресует указатель.

К базовым операциям при работе со связными структурами относятся: добавление нового элемента, чтение элемента и удаление элемента.

При этом необходимо заметить, что операции чтения и удаления возможны только для непустых линейных структур, поэтому обращению к этим процедурам должна предшествовать проверка пустоты стека (очереди, списка).

Основные различия линейных динамических структур состоит в том, что включение, исключение и доступ к элементам осуществляется по-разному.

4.4.1. Стек

Стеком называется упорядоченный набор элементов, в котором размещение новых элементов и удаление существующих осуществляется только с одного конца, называемого вершиной стека. В любой момент времени доступен лишь один элемент - верхний. Стеки довольно часто встречаются в практической жизни: детская пирамидка, стопка тарелок.

Организация данных в виде стека широко используется в вычислительной технике; организация модульного программирования (процесс входов и выходов в подпрограммы, вложенных циклов), организация вычислений (соблюдение приоритетов при выполнении операций) и т.д.

Рассмотрим операции над стеком.

Дополнение стека новым элементом:

```
Procedure DOPOLNEN ( Var S : PSTRC; DAT : integer );
  Var NOV : PSTRC;        // <- указатель нового элемента стека
Begin  NEW ( NOV );       // <- запрос ОП для нового элемента
       NOV^.INF := DAT;
       NOV^.PTR := S;
       S := NOV;
End;
```

В процедуре дополнения используются два формальных параметра: переменная S, которая является указателем на вершину (стековая переменная) и переменная DAT, которая содержит значение, заносимое в новый элемент стека. Необходимо помнить, что в процессе работы значение S меняется, поэтому необходимо обеспечивать возвращение этой переменной в вызывающую процедуру (в данном случае используя параметр-переменную). В данной процедуре принципиальна последовательность операторов. Если происходит дополнение пустого стека, то переменная S предварительно должна быть обнулена:

(S := NIL).

Удаление последнего элемента из стека:

```
Procedure UDALENIE ( Var S : PSTRC);
Var OLD : PSTRC;  // <- указатель на удаляемый элемент
Begin If S <> NIL then // <- есть объект для удаления
  begin
       OLD := S;
       S := S^.PTR;
       Dispose (OLD);
  end
end;
```

В процедуре удаления используется только один формальный параметр – вершина стека, которая и является удаляемым элементом. Собственно удаление выполняет оператор S := S^.PTR, который фактически переставляет указатель вершины стека на предыдущий элемент. Для освобождения памяти (удаления из памяти элемента) используется переменная OLD. И в данном случае последовательность операций принципиальна.

Чтение последнего элемента из стека:

```
Function CHTENIE ( S : PSTRC) : integer;
Begin       If S <> NIL then
       CHTENIE := S^.INF;
end;
```

Под чтением в самом общем смысле понимается обращение к полю данных вершины стека, т.е. использование S^.INF в операторе, например, присваивания или вывода, уже и будет чтением стека.

Одной из наиболее интересных задач, решаемых с использованием стека, является перевод арифметических выражений из инфиксной формы в постфиксную или префиксную. Или наоборот из постфиксной или префиксной в инфиксную.

Выражение записано в инфиксной форме, когда знак операции записан между операндами (например, A+B или A/B); в постфиксной – знак за операндами (AB+ или AB/), в префиксной – перед операндами (+AB или /AB). При этом префиксная и постфиксная форма не являются зеркальным отражением друг друга (только в очень редких случаях при

использовании примитивных выражений). Используется такой перевод в компиляторах систем программирования для удаления скобок при организации вычислений.

Рассмотрим несколько примеров преобразования:

Инфиксное представление	Постфиксное представление	Префиксное представление
A+B-C	AB+C-	-+ABC
(A-B)*(C-D)	AB+CD-*	*+AB-CD
A*B/C-D+E/F/(G+H)	AB*C/D-EF/GH+/+	+-/*ABCD//EF+GH

Рассмотрим алгоритм перевода выражения из префиксной формы в инфиксную. Для упрощения алгоритма будем считать, что операндами выражения являются символы (символьные переменные), знаки записываются без пробелов. Таким образом, из алгоритма возможно исключить все не относящиеся к сути проблемы детали.

Из тех же соображений в данной программе не предусмотрен контроль ошибок. Т.е. мы предполагаем, что количество операндов на один больше, чем знаков операций.

В качестве исходных данных может использоваться, например, следующее арифметическое выражение: +-/+AB+CD/-EFG*KL. При этом результатом является печать выражения в инфиксной форме:

(((A+B)/(C+D))-((E-F)/G))+(K*L).

В этой схеме используются две стековые переменные P1 и P2. Стек P1 – основной, стек P2 – рабочий для формирования промежуточных операндов. Предполагается, что мы используем арифметические операции двух приоритетов, так же как в математике сложение и вычитание имеют приоритет ниже, умножение и деление. Выражения, в которых операции имеют более низкий приоритет, в результате взяты в скобки. Избавление от лишних скобок несложно, для чего также удобно использовать стек, но эта уже следующая задача. Можно организовать специальную процедуру и вызвать ее перед блоком «Записать выражение в файл результатов».

Обработка строки (выражения) начинается с первого знака операции, за которым следуют два операнда, которые последовательно достаются из стека. Из них формируется строковое выражение, которое в свою очередь является операндом следующей операции (либо результатом), заносится в стек и используется при формировании следующего выражения и т.д.

Несмотря на то, что в начале обработки в стек заносится символ, элементом стека является строка, в итоге равная по длине исходному выражению, но необходимо учесть при объявлении стека.

В тексте программы, написанной на основе данного алгоритма, опущены процедуры дополнения и удаления стека, т.к. они аналогичны рассмотренным ранее, а процедура чтения не используется.

```
Program stack2;
type Pst = ^st;                    // - указатель на запись
       r = string[40];
     st = record
            op : r;                // - хранимый операнд
            Ps : Pst;              // - указатель на следующую запись
           end;
var      fi,                       // - файл  с исходными данными
         fr : text;                // - файл для результатов
          s : r;
    P1, P2 : Pst;                  // - указатели на стеки
{ Множество операций: } const oper : set of char = ['/', '*', '+', '-'];
{ Процедуры дополнения и удаления }
Procedure DOPOLNEN ( Var S : PST; DAT : r );
```

```
  Var NOV : PST;   // <- указатель нового элемента стека
Begin  NEW ( NOV );      // <- запрос ОП для нового элемента
       NOV^.op := DAT;
       NOV^.Ps := S;
       S := NOV;
End;

Procedure UDALENIE ( Var S : PST);
Var OLD : PST;    // <- указатель на удаляемый элемент
Begin If S <> NIL then // - есть объект для удаления
  begin
       OLD := S;
       S := S^.Ps;
       Dispose (OLD);
  end
end;

begin
 assign(fi,  'steck1.txt'); reset(fi);
 assign(fr,  'steck2.txt');  rewrite(fr);
 P1 := nil;
 read(fi, s);  writeln(fr, s);  close(fi);
 for var i: byte := length(s) downto 1 do  // - запись выражения в стек
           DOPOLNEN (P1, s[i]);           //  посимвольно в "прямом" порядке
 repeat
         repeat
   s := P1^.op;                          //- запоминание операции
 //  UDALENIE(P1);                       // - проверка следующего символа:
   if (P1^.op[1] in oper)or(P1 = nil) then DOPOLNEN (P2, s)
   else begin                            // - если не операция, то
         Insert('(' + P1^.op, s, 0);     //   составляем из операндов
         UDALENIE (P1);                  //  выражение и добавляем его
         s := s + P1^.op + ')';          //  в стек P2
         DOPOLNEN(P2, s);  UDALENIE (P1);
        end;
  until P1 = nil;               // - просмотрели все выражение
  while P2 <> nil do            // - перепись составленных выражений
   begin                        //   в стек P1 в "прямом" порядке
    DOPOLNEN (P1, P2^.op);  UDALENIE (P2);
   end;
 until P1^.Ps = nil;            // - в стеке P1 - один элемент
 writeln(fr, 'Арифметическое выражение в инфиксной форме:');
for var i: byte := 2 to length(P1^.op) - 1 do  write(fr, P1^.op[i]);
 UDALENIE (P1); close(fr);
 end.
```

4.4.2. Очередь

Очередью называется упорядоченный набор элементов, которые могут удаляться с одного ее конца (называемого началом очереди) и помещаться в другой конец этого набора (называемого концом очереди).

При организации очереди доступны могут быть две ее позиции: начало и конец. И в этом случае для работы с очередью необходимо описать две переменные. Однако для обработки должен быть доступен только элемент, поступивший раньше остальных. Поэтому во избежание ошибок можно ограничиться только одной переменной для организации очереди. В этом случае несколько усложняется операция дополнения очереди. В случае двух переменных работа с очередью практически аналогична работе со стеком.

Рассмотрим операции над очередью при условии, что доступна только одна ее позиция – начало очереди. Дополнение элементов и в этом случае происходит в ее конец.

Дополнение очереди новым элементом:

```
Procedure DOPOLNEN ( Var Q : PSTRC; DAT : integer );
Var NOV, CUR, PR : PSTRC;
Begin  NEW ( NOV );      // <- запрос ОП для нового элемента
       CUR := Q; PR := NIL;
       NOV^.INF := DAT;
       NOV^.PTR := NIL;
       WHILE CUR <> NIL do begin     // <- движение вдоль очереди
                    PR := CUR; CUR := CUR^.PTR;
                              end;
//  Подключение нового элемента очереди:
      If PR = NIL then Q := NOV            { <- новый элемент - первый     }
            else PR^.PTR := NOV;     { <- новый элемент - очередной }
end;
```

При дополнении очереди необходимо аналогично дополнению стека выделить память под новый элемент и сформировать его. Для добавления его к очереди требуется пройти вдоль очереди до ее конца. Переменная CUR используется как для обнаружения конца очереди (CUR <> NIL), так и для движения вдоль очереди (CUR := CUR^.PTR). Переменная PR используется для подключения к очереди нового элемента, а в первую очередь для определения, является ли добавляемый элемент первым (т.е. после добавления он будет единственным) или нет. Добавление происходит по-разному. Первый становится вершиной очереди, а в другом случае на него указывает последний элемент очереди (PR^.PTR := NOV), после чего он становится последним.

Чтение элемента очереди (первого):

Чтение очереди ничем не отличается от чтения стека. Также в реальном программировании редко используется в виде отдельной процедуры или функции, а при наличии процедуры ее можно использовать как для чтения очереди, так и для чтения стека при совпадении типов поля данных.

```
Function CHTENIE ( Q : PSTRC) : integer;
BEGIN  If Q <> NIL then
         CHTENIE := Q^.INF  // возврат значения начала очереди
END;
```

Удаление элемента из очереди (первого):

```
Procedure UDALENIE ( Var Q : PSTRC);
   Var OLD : PSTRC;
Begin  OLD := Q;
       If Q <> NIL then begin     // есть объект для удаления
             Q := Q^.PTR; Dispose (OLD);
                        end
end;
```

Процедура удаления также аналогична удалению из стека.

4.4.3. Список

Список - это соединение элементов в порядке убывания или возрастания ключевого признака - одного из элементов записи, из которых состоит список. Каждый элемент списка может быть сложной структурой. Для целого ряда задач используется неупорядоченный список, создание которого идентично созданию очереди. Примем соглашение, что каждый

такой случай оговаривается отдельно. В общем случае элементы списка упорядочены по возрастанию.

Основное отличие списка от стека и очереди заключается в том, что любой элемент списка может быть доступен для обработки. При этом необходимость в удалении не возникает.

Аналогично очереди, при организации списка могут использоваться как две переменные: указатели на начало и конец списка, так и одна.

Список может быть однонаправленным и двунаправленным.

Рассмотрим операции над списком. Кроме базовых операций к списку можно также применить следующие операции:

- поиск и вывод заданного элемента;
- чтение и вывод всех элементов списка;
- уничтожение списка;

При этом добавление нового элемента происходит в произвольное место списка, а также удаление из списка возможно любого элемента, заданного поисковым признаком.

Дополнение списка новой записью:

```
Procedure DOP ( VAR S : PSTRC; DAT : integer );
    Var NOV, CUR, PR : PSTRC;  KEY : integer;
Begin
    CUR := S; PR := NIL;  KEY := DAT;  Writeln( ' KEY = ', KEY);
    NEW ( NOV );  NOV^ := DAT;
    While (CUR <> NIL) and (CUR^.INF < KEY) do begin
             PR := CUR; CUR := CUR^.PTR; end; // <- Движение вдоль списка
    NOV^.PTR := CUR;     // <- подключение нового элемента справа
  //   Подключение нового элемента слева:
    If PR = NIL then S := NOV
                else PR^.PTR := NOV
end;
```

Процедура дополнения практически похожа на дополнение очереди, но имеет принципиальные отличия. Движение вдоль списка происходит не до конца, а до момента обнаружения места элемента (проверка по ключу CUR^.INF < KEY). В данном случае проход до конца списка является частным случаем. И в начало списка элемент может попасть не один раз, а несколько.

Чтение списка:

```
Procedure CHTENIE ( S : PSTRC );
    Var CUR : PSTRC;
Begin   CUR := S;
    While CUR <> NIL do begin // <- движение вдоль списка
          Writeln (CUR^.INF);
          CUR := CUR^.PTR;
                      end
end;
```

При чтении списка используется дополнительная переменная CUR, в данном случае которую можно не использовать. Она нужна в том случае, если необходимо сохранить значение начала списка, которое в результате этой операции будет утрачено. Учитывая, что в процедуру передано значение, а не адрес переменной S, для дальнейшей работы программы это угрозы не представляет.

Поиск в списке по заданному значению ключа:

```
procedure POISK ( S : PSTRC; KEY : integer );
    Var CUR : PSTRC;
Begin
    Writeln ( 'поисковый ключ = ', KEY );  CUR := S;
    While (CUR <>  NIL) and (CUR^.INF <> KEY) do CUR := CUR^.PTR;
    If CUR = NIL then Writeln (' ПОИСК НЕУСПЕШЕН')
                 else  Writeln(CUR^.INF); // <- вывод найденного элемента
end;
```

То же самое про переменную S можно сказать и в этом случае. Поиск может использоваться не только для вывода элемента, но и для другой обработки. В самом общем случае алгоритм поиска можно рассматривать как чтение элемента с предварительным обнаружением.

Удаление из списка записи с заданным значением ключа.

```
procedure UD ( Var S : PSTRC; KEY : integer );
    Var CUR, PR : PSTRC;
Begin  CUR := S; PR := NIL; Writeln ( 'удаление записи = ', KEY);
       While (CUR <>  NIL) AND (CUR^.INF <> KEY) do begin
             PR := CUR; CUR := CUR^.PTR;     end; //<- движение вдоль списка
       If CUR = NIL then begin
                    Writeln ( 'запись не найдена'); Exit end;
       If PR = NIL then begin Writeln('удаляем первый элемент ');
                              P (CUR^);   S := CUR^.PTR;  end
                   else begin Writeln ('удаляем не первый элемент');
                              P (CUR^);   PR^.PTR := CUR^.PTR;  end;
       Dispose (CUR);    // <- освобождение ОП
end;
```

При удалении элемента, в том случае если обнаружен элемент для удаления, можно выделить два случая: когда удаляемый элемент первый (меняется значение указателя на начало списка S := CUR^.PTR) и не первый, когда связываются между собой предыдущий и последующий элементы от текущего (PR^.PTR := CUR^.PTR).

Рассмотрим, что изменится при работе со списком, если он двунаправленный.

Объявление двунаправленного списка:

```
Type PSTRC = ^ STRC;
     STRC = record
        INF : integer;   { <- информация элемента списка }
        PTR : PSTRC;     { <- указатель на предыдущий элемент }
        NXT : PSTRC      { <- указатель на следующий элемент }
              end;
```

```
Var S: PSTRC;
```

Дополнение двунаправленного списка:

```
procedure DOPOLNEN (var q : Pstrc; key : real);
var nov, cur, pr : Pstrc;

begin
 cur := q; pr := nil;
 new(nov);           // <-  выделение памяти для нового элемента
 nov^.d := key;
```

```
 while (cur <> nil)and(cur^.d < key) do //<- движение вдоль списка
       begin    pr := cur; cur := cur^.Nxt;  end;
 nov^.Nxt := cur;   // <-  подключение нового элемента
 nov^.Ptr := pr;                          // <-  к соседним;
 if pr = nil then begin q^.Ptr := nov;// <-  новый элемент - первый
                        q := nov;
                  end
          else begin  pr^.Nxt := nov;// <-новый элемент - очередной
                         cur^.Ptr := nov;
                  end;
end;
```

Удаление элемента из двунаправленного списка:

```
procedure del(var q : Pstrc);
var old : Pstrc;
begin
 if q <> nil then begin        // <- список не пуст
          repeat
              old := q;
              q := q^.Nxt;      // <- переход на следующий элемент
              dispose(old);     // <- удаление текущего элемента
              q^.Ptr := nil;
          until q^.Nxt = nil;
                    dispose(q);  // <- удаление последнего элемента
                  end
              else writeln(fr, 'Список пуст. Удаление списка
невозможно.');
end;
```

Чтение двунаправленного списка возможно в любом направлении. При этом, если имеются указатели на начало и конец, то это просто две одинаковые процедуры, все отличие которых заключается в используемом указателе

(CUR := CUR^.PTR или CUR := CUR^. NXT).

Если же используется только указатель на начало списка, то его нужно сначала пройти в одном направлении, а только затем – в обратном:

```
Procedure CHTENIE ( S : PSTRC );
    Var CUR : PSTRC;
Begin   CUR := S;
        While CUR <> NIL do begin // <- движение вдоль списка без печати
                    CUR := CUR^.PTR; end;
        While CUR <> NIL do begin // <- движение вдоль списка обратно
                    Writeln (CUR^.INF);
                    CUR := CUR^.Nxt; end;
end;
```

4.5. Пример выполнения работы

Задание. Дан текстовый файл, содержащий целые числа. Переписать в другой текстовый файл сначала все отрицательные числа в порядке поступления, а затем остальные в порядке, обратном поступлению.

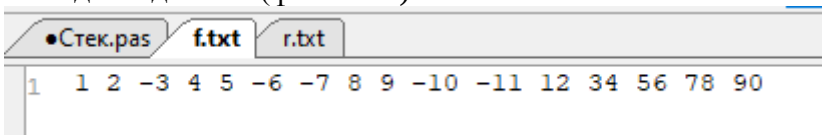
Данная задача слишком далека от реальных задач, решаемых при программировании, и носит исключительно иллюстративный характер.

Использование файлов определено условиями задачи, хотя ввод данных вполне мог бы быть осуществлен и с экрана. Файл для вывода результатов – обязателен.

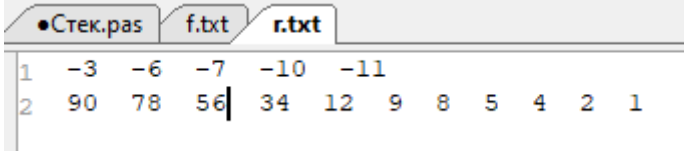
Кстати, если бы обе группы нужно было выводить в порядке, обратном поступлению, то нужно было бы организовать два стека и выводить их по очереди.

```
Type PSTRC = ^ STRC;
STRC = record
INF : integer;     // <- информация элемента
PTR : PSTRC    // <- указатель на предыдущий элемент
      end;

Var S : PSTRC;
    a : integer;
    F1, F2 : text;

Procedure DOPOLNEN ( Var S : PSTRC; DAT : integer );
  Var NOV : PSTRC;        // <- указатель нового элемента стека
Begin  NEW ( NOV );       // <- запрос ОП для нового элемента
       NOV^.INF := DAT;
       NOV^.PTR := S;
       S := NOV;
End;

Procedure UDALENIE ( Var S : PSTRC);
Var OLD : PSTRC;  // <- указатель на удаляемый элемент
Begin If S <> NIL then // - есть объект для удаления
  begin
       OLD := S;
       S := S^.PTR;
       Dispose (OLD);
  end
end;

Function CHTENIE ( S : PSTRC) : integer;
Begin       If S <> NIL then
       CHTENIE := S^.INF;
end;

Begin
      Assign (f1, 'f.txt'); Reset (F1);
      Assign (f2, 'r.txt'); Rewrite (F2);
      S:=Nil;
      While not eof (F1) do begin
            Read (f1, a); if  a<0 then write (F2, a, '  ' )
                                  Else DOPOLNEN (S, a);
                            end;
                            writeln (F2);
      While S <> NIL do begin
      Write (F2, CHTENIE (S), '  '); UDALENIE (S);
                  end;
Close (F1); Close (F2);
End.
```

Исходные данные (файл f.txt):



Результат работы программы представлен на листинге далее.



4.6. Варианты на выполнение работы №11

1. Реализовать два однонаправленных линейных списка, элементы в которых целого типа. Сформировать список из элементов, которые входят в оба списка.

2. Реализовать стек, содержащий целые числа (типа byte). Напечатать порядке, обратном поступлению, все числа, имеющие в своем составе заданную цифру, а затем все остальные числа также в порядке, обратном поступлению. (Использовать дополнительный стек).
3. Реализовать два однонаправленных линейных списка, элементы в которых вещественного типа. Сравнить эти списки (совпадают ли они по количеству элементов в них), а также сравнить средние значения элементов этих списков, а также максимальные значения.
4. Реализовать стек, содержащий массивы пяти целых чисел. Напечатать в обратном порядке суммы всех массивов, а произведения – в порядке поступления (используя второй стек).
5. Реализовать два однонаправленных линейных списка, элементы в которых целого типа. Сформировать список из элементов, которые входят только в один список.
6. Для заданного арифметического выражения со скобками одного типа напечатать операции в порядке их выполнения. Использовать стек.
7. Реализовать стек, содержащий массивы трех вещественных чисел. Напечатать в обратном порядке минимальные значения всех массивов, а максимальные – в порядке поступления (используя второй стек).
8. Реализовать стек, содержащий целые числа. Напечатать в порядке, обратном поступлению, все числа, меньшие заданного. Сформировать очередь, содержащую произведения двух соседних элементов в первом стеке чисел.
9. Содержимое текстового файла S, разделенное на строки, переписать в текстовый файл Q, перенося при этом в конец каждой строки все входящие в нее цифры (с сохранением взаимного исходного порядка как среди цифр, так и среди остальных символов). Использовать очередь.
10. Реализовать два однонаправленных списка, элементы в которых строкового типа. Определить количество слов, встречающихся в двух списках. Сформировать из них новый список.
11. Реализовать очередь, содержащую строки (состоящие из букв русского алфавита). Напечатать строки в порядке поступления. Определить количество строк, начинающихся на ту же букву, которой заканчивается предыдущая. Сформировать стек из таких строк и напечатать в порядке, обратном поступлению.
12. Содержимое текстового файла S, разделенное на строки, переписать в текстовый файл Q, перенося при этом в конец каждой строки все входящие в нее цифры (в порядке, обратном исходному как среди цифр, так и среди остальных символов). Использовать стек.
13. Реализовать два однонаправленных линейных списка, элементы в которых целого типа. Сцепить эти списки, определить количество элементов в них, а также удвоить каждое вхождение заданных элементов в новый список элементов.
14. Реализовать стек, содержащий записи, состоящие из двух полей: имени и фамилии. Напечатать сначала в порядке обратном поступлению фамилии всех людей с заданным именем, а затем остальные элементы стека. Определить количество элементов с заданным именем и остальных.
15. Реализовать стек, содержащий массивы трех символов. Напечатать в обратном порядке массивы, элементы в которых упорядочены по убыванию, а остальные массивы – в порядке поступления (используя второй стек).
16. Реализовать стек, содержащий массивы трех строк. Проверить каждую строку, расположены ли в них символы в порядке возрастания. Вывести сначала такие массивы в

порядке, обратном поступлению, в которых все строки отвечают этому требованию. Остальные массивы - в порядке поступления. (Использовать второй стек).

17. Реализовать два двунаправленных линейных списка, элементы в которых вещественного типа. Определить количество элементов в них. Сцепить эти списки (упорядочив), а также напечатать отрицательные элементы нового списка.
18. Реализовать стек, содержащий символы латинского алфавита. Проверить, расположены ли они в алфавитном порядке. Если да, то напечатать их в порядке, обратном поступлению, если нет, то в порядке поступления.
19. За один просмотр файла типа «массив трех вещественных чисел» напечатать (используя очередь) сначала все отрицательные произведения элементов массивов, а затем все положительные, сохраняя при этом порядок поступления массивов.
20. Реализовать с помощью однонаправленного линейного списка операцию умножения длинного целого числа на N (целую цифру).
21. Реализовать с помощью двунаправленного линейного списка операцию вычитания двух длинных целых чисел.
22. Реализовать двунаправленный список, элементы в котором строкового типа. Сформировать новый список из элементов первого списка, длина которых меньше заданной.

Задания повышенной сложности:

1. Реализовать программу вычисления арифметического выражения со скобками с операциями двух приоритетов.
2. Реализовать программу перевода выражений из инфиксной формы в префиксную.
3. Реализовать программу перевода выражений из инфиксной формы в постфиксную.
4. Реализовать двунаправленный линейный список, элементы в котором строкового типа. Удалять из данного списка каждый к-й элемент, начиная с первого, перемещаясь по кругу, печатать каждый удаляемый элемент, до полного освобождения списка.

4.7. Контрольные вопросы

1. Что относится к динамическим структурам данных и что это такое?
2. Что такое линейная структура?
3. Перечислите свойства линейных структур?
4. Как организовать данные линейной структуры с помощью с помощью динамических переменных (указателей)?
5. Что такое стек, какие базовые операции применяются при работе со стеком?
6. Что такое инфиксная форма для записи арифметических выражений? Префиксная? Постфиксная?
7. Выполните перевод из инфиксной формы в префиксную и постфиксную следующих выражений:
   - A-D/F/S+(R+G*L)
   - (F+G+H)*(Y+X*S)/X

– F+G+H+J+E+S*W-5

8. Что такое очередь, какие базовые операции применяются при работе с очередью?
9. Что такое список, какие базовые операции применяются при работе со списком?
10. Объясните особенности организации с помощью динамических переменных однонаправленного и двунаправленного списка.

5. Лабораторная работа №12

Разработка программы для работы с объектами

5.1. Цель работы

Целью лабораторной работы является освоение:

- Назначения и строения объектов;
- Назначения и состава полей и методов объекта;
- Правил объявления и использования полей объектов;
- Правил написания и переопределения статических методов объектов;
- Вызова методов объектов;
- Определения области действия имен объектов;
- Правил создания, вызова и уничтожения динамических объектов.

5.2. Задание на выполнение работы

1. Разработать программу с использованием статических (объекта предка и объекта потомка), динамического объекта и статических методов. Вариант задания – лабораторная работа №1 (задание 1) третьего семестра (обработка двумерного динамического массива).
2. Разработать программу с использованием статических (объекта предка и объекта потомка), динамического объекта и статических методов. Вариант задания – п.5.6 настоящего пособия.
3. Разработать программу задания №2 с использованием виртуального метода.

5.3. Требования к выполнению

- Продемонстрировать полиморфизм на примере одного из методов (по выбору студента).
- Работоспособность программ продемонстрировать на нескольких комплектах исходных данных.
- Названия классов должны иметь «говорящие» имена.

5.4. Краткие теоретические сведения

Объектно-ориентированное программирование (ООП) – современное направление развития программирования, которое реализовано в той или иной степени во всех современных системах программирования.

ООП характеризуется тремя основными свойствами: инкапсуляцией, наследованием и полиморфизмом.

Инкапсуляция – объединение в одном объекте данных (полей) и действий над ними (методов).

Наследование – возможность использования уже определенных ранее объектов, что позволяет создавать иерархию объектов. Эта иерархия может иметь сложную структуру.

Полиморфизм – возможность определения единого по имени действия (процедуры или функции), применимого ко всем объектам иерархии наследования; причем каждый объект иерархии может иметь особенность реализации этого действия.

Объектно-ориентированный подход имеет ряд важных аспектов:

- модульность, позволяющую локализовать область действия подпрограмм и переменных и дающую возможность изменять локальные переменные, не изменяя других программных модулей;
- абстракция данных (определение абстрактного типа);
- динамическая связка подпрограмм программы (при изменении отдельного модуля можно не выполнять компиляцию всей программы);
- наследование (порожденный класс наследует все свойства порождающего класса, позволяя в то же время его расширить).

Описание классов.

Класс - это составной тип, состоящий из полей (переменных), методов (процедур и функций) и свойств.

Описание класса имеет вид:

```
type
  имя класса = class
    секция1
    секция2
      ...
  end;
```

Каждая секция имеет вид:

```
Модификатор доступа

 описания полей
 объявления или описания методов и описания свойств
```

Модификатор доступа, определяющий область видимости, в первой секции может отсутствовать, при этом подразумевается модификатор internal (видимость всюду внутри сборки).

Методы могут описываться как внутри, так и вне класса. При описании метода вне класса его имя предваряется именем класса с последующей точкой.

Например:

```
type
  Person = class
  private
    fName: string;
    fAge: integer;
  public
    constructor Create(Name: string; Age: integer);
    begin
      fName := Name;
      fAge := Age;
    end;
    procedure Print;
    property Name: string;
    property Age: integer;
```

```
  end;

procedure Person.Print;
begin
  Writeln($'Имя: {Name} Возраст: {Age}');
end;
```

После слова ***class*** в скобках может быть указано имя класса-предка.

Перед словом ***class*** может быть указано ключевое слово ***sealed*** – в этом случае от класса запрещено наследовать.

Все описания и объявления внутри класса образуют ***тело класса***. Поля и методы образуют ***интерфейс*** класса.

Классы могут описываться только на глобальном уровне. Локальные определения классов (т.е. определения в разделе описания подпрограмм) запрещены.

Переменные типа класс.

В языке PascalABC.NET классы являются ссылочными типами. Это значит, что переменная типа класс хранит в действительности ссылку на объект.

Переменные типа класс называются ***объектами*** или ***экземплярами класса***. Они инициализируются вызовом конструктора класса - специального метода, выделяющего память под объект класса и инициализирующего его поля:

```
var p: Person := new Person('Иванов',20);
```

После инициализации через переменную типа класс можно обращаться к публичным членам класса (полям, методам, свойствам), используя точечную нотацию (составной идентификатор):

```
Print(p.Name,p.Age);
p.Print;
```

Вывод переменной типа класс.

По умолчанию процедура Write для переменной типа класс выводит содержимое её публичных полей и свойств в круглых скобках через запятую:

```
Write(p); // (Иванов, 20)
```

Чтобы изменить это поведение, в классе следует виртуальный метод ToString класса Object переопределить - в этом случае именно он будет вызываться при выводе объекта.

Например:

```
type
  Person = class
    ...
    function ToString: string; override;
    begin
      Result := $'Имя: {Name} Возраст: {Age}';
    end;
  end;
  ...
var p: Person := new Person('Иванов',20);
Writeln(p); // Имя: Иванов  Возраст: 20
```

Конструкторы.

Объекты создаются с помощью специальных методов, называемых ***конструкторами***.

Конструктор представляет собой функцию, создающую объект в динамической памяти, инициализирующую его поля и возвращающую ссылку на созданный объект. Эта ссылка обычно сразу присваивается переменной типа класс. При описании конструктора

вместо ключевого слова ***procedure*** используется ключевое слово ***constructor***. Кроме того, для конструктора не указывается тип возвращаемого значения.

Например:

```
type
  Person = class
  private
    nm: string;
    ag: integer;
  public
    constructor Create(name: string; age: integer);
  end;
...
constructor Person.Create(name: string; age: integer);
begin
  nm := name;
  ag := age;
end;
```

В PascalABC.NET конструктор всегда должен иметь имя ***Create***. При описании конструктора внутри класса можно опускать его имя:

```
type
  Person = class
    constructor (name: string; age: integer);
    begin
      nm := name;
      ag := age;
    end;
  end;
```

В силу особенностей реализации вызовов конструкторов в .NET в PascalABC.NET всегда создается конструктор без параметров (независимо от того, определен ли другой конструктор, а в классе может создаваться несколько конструкторов). Этот конструктор инициализирует все поля нулевыми значениями (строковые поля - пустыми строками, логические - значением False).

Для вызова конструктора можно использовать два способа.

1 способ. В стиле Object Pascal.

Для вызова конструктора следует указать имя класса, за которым следует точка-разделитель, имя конструктора и список параметров.

Например:

```
var p: Person;
p := Person.Create('Иванов',20);
```

2 способ. С помощью операции `new` (предпочтительный).

```
var p: Person;
p := new Person('Иванов',20);
```

Операция new.

Операция ***new*** имеет вид:

new *ИмяКласса* (*ПараметрыКонструктора*)

Она вызывает конструктор класса ***ИмяКласса*** и возвращает созданный объект. Например:

```
type
  My = class
    constructor Create(i: integer);
    begin
    end;
  end;
var m: My := new My(5);
```

Деструкторы.

Деструктор в Object Pascal - специальная процедура, уничтожающая объект и освобождающая динамическую память, которую этот объект занимал. При описании деструктора вместо служебного слова ***procedure*** используется служебное слово ***destructor***. Например:

```
destructor Destroy;
begin
  ...
end;
```

Присваивание и передача в качестве параметров подпрограмм.

Переменная типа `class` является ссылкой и хранит ссылку на объект, создаваемый вызовом конструктора.

Как ссылка переменная типа `class` может хранить значение **nil**:

```
p := nil;
...
if p = nil then ...
```

При присваивании переменных типа `class` копируется только ссылка. После присваивания обе переменные типа `class` будут ссылаться на один объект и совместно модифицировать его:

```
var p1,p2: Person;
...
p1 := new Person('Петров',20);
p2 := p1;
p1.IncAge;
p2.Print; // Имя: Петров  Возраст: 21
```

Статические классы, поля, методы, свойства и конструкторы.

В классе можно объявить так называемые статические поля, свойства и методы. Они не принадлежат конкретному экземпляру класса, а связаны с классом. Для их вызова используется точечная нотация, причем, перед точкой используется не имя объекта, а имя класса. Чтобы поле или свойство или метод сделать статическим, перед его именем следует указать ключевое слово ***static***. При описании статических свойств в секциях read и write можно указывать только статические поля или методы.

Например, определим для класса Person количество созданных объектов этого класса как статическое поле и организуем доступ к этому полю на чтение с помощью статической функции. После каждого вызова конструктора значение статического поля будет увеличиваться на 1:

```
type
  Person = class
  private
    name: string;
    age: integer;
    static cnt: integer := 0;
  public
    static property Coun: integer read cnt;
    constructor (n: string; a: integer);
    begin
      cnt := cnt + 1;
      name := n;
      age := a;
    end;
    static function Count: integer;
    begin
      Result := cnt;
    end;
  end;

begin
  var p: Person := new Person('Иванов',20);
  var p1: Person := new Person('Петров',18);
  Writeln(Person.Count); // обращение к классовому методу Count
end.
```

В отличие от статических полей и методов, обычные поля и методы называются экземплярными. Из обычных методов можно обращаться к экземплярным и статическим полям, но из статических методов можно обращаться только к статическим полям.

Аналогично можно определить также статический конструктор, предназначенный для автоматической инициализации классовых полей. Статический конструктор описывается с ключевым словом static и гарантированно вызывается перед вызовом любого статического метода и созданием первого объекта этого класса.

Например, определим в классе Person статическое поле - массив объектов типа Person - и инициализируем его в статическом конструкторе. Потом указанный массив можно использовать в реализации статической функции RandomPerson, возвращающей случайный объект типа Person:

```
type
  Person = class
  private
    static arr: array of Person;
    name: string;
    age: integer;
  public
    static constructor;
    begin
      SetLength(arr,3);
      arr[0] := new Person('Иванов',20);
      arr[1] := new Person('Петрова',19);
      arr[2] := new Person('Попов',35);
    end;
    //...
    static function RandomPerson: Person;
    begin
      Result := arr[Random(3)];
    end;
  end;
const cnt = 10;
begin
```

```
  var a := new Person[cnt];
  for var i:=0 to a.Length-1 do
    a[i] := Person.RandomPerson;
end.
```

Класс также можно описывать как статический:

```
type
  MyStatic = static class
    static Pi: real := 3.14;
    static function Pi2 := Pi * Pi;
  end;
```

В этом случает все его методы, поля, свойства и конструкторы должны быть статическими. Запрещается создавать экземпляры статических классов. Кроме того, от статических классов нельзя наследовать и статический класс не может быть предком.

5.5. Пример выполнения работы

Рассмотрим пример выполнения первого задания для статического массива, так как для рассматриваемой темы это не имеет большого значения.

Программа:

```
     {  Статические методы.}
program laborat5;
{$F+}
Const m = 5; n = 6;
Type TA = array[1..m,1..n] of real;
Type obj = class //object
          A : TA;
          B : array[1..3,1..3] of real;
          sum, min, max : real;
          imin, imax, jmin, jmax, kol: integer;
          Procedure vvod (fn : string);
          Procedure P;
          Procedure obr;
private
          Procedure Vyvod;
             end;

      ob1 = class (obj)
       sred:real;
       Procedure obr;
       end;

{---------- Методы объекта----------------}
{---------- Ввод исходных данных  ----------------}
Procedure obj.vvod(fn : string);
var i,j : byte;
      Fi : text;
Begin
Assign(Fi, Fn); Reset(FI);  Writeln;
For i:=1 to m do
      for j:=1 to n do read(FI, A[i,j]);
close(FI);
end;

{------------- Вывод исходных данных ------------}
Procedure obj.P;
Var i, j : integer;
Begin
For i:=1 to m do
```

```
    for j:=1 to n do case j of
                         1, 3, 5 : write(' ':2, A[i,j] :10);
                            2, 4 : write(' ':2, A[i,j] :8:2);
                               6 : writeln(' ':2, A[i,j] :10:1);
                       end;

end;

{----------- Обработка данных -----------}
Procedure obj.obr;
Var   R : real;
     i, j, jn: integer;
Label MET;
Begin
sum:=0; kol:=0;
For j:=1 to n div(2) do begin
        for i:=1 to m do if A[i,2*j]>0 then begin B[1,j]:=A[i,2*j];
                                                  B[2,j]:=i; B[3,j]:=2*j;
                                                  sum:=sum + B[1,j];
                                                  kol:=kol + 1; jn:=j; goto MET;
                                                  end;
        B[1,j]:=0; B[2,j]:=0; B[3,j]:=0;
                         MET:end;
min:=B[1,jn]; max:=B[1,jn]; imin:=1; imax:=1; jmin:=jn; jmax:=jn;
For j:=1 to n div(2) do if B[2,j]>0 then if B[1,j]>max then
                                    begin max:=B[1,j]; imax:=round(B[2,j]);
                                    jmax:=round(B[3,j]); end
                                      else if B[1,j]<min then
                                      begin min:=B[1,j]; imin:=round(B[2,j]);
                                              jmin:=round(B[3,j]); end;
If imin<>imax then for j:=1 to n do begin
                    R:=A[imin,j]; A[imin,j]:=A[imax,j]; A[imax,j]:=R;
                                    end;
end;

{---------- Вывод данных ------------}
Procedure obj.Vyvod;
Var  //sum, min, max : real;
     i, j: integer;

Begin
Writeln(' Сформированный массив : ');
For i:=1 to 3 do begin
    for j:=1 to n div(2) do if i=1 then write(B[i,j]:7:1)
                                   else write(B[i,j]:7:0);
  writeln;       end;

Writeln(#10#13,' Полученные результаты : ', #10#13,
                ' sum = ', sum :6:2,'   kol = ', kol :2,
                 #10#13,' min = ', min:6:2,'':4,'max = ', max:6:2,
                 #10#13,' imin = ', imin:1,'':7,'imax = ', imax:1,
                 #10#13,' jmin = ', jmin:1,'':7,'jmax = ',jmax:1, #10#13);
end;

{----------- Обработка данных потомка -----------}
Procedure ob1.obr;
Var   R : real;
     i, j, jn: integer;
Label MET;
Begin
sum:=0; kol:=0;
For j:=1 to n div(2) do begin
        for i:=1 to m do if A[i,2*j]>0 then begin B[1,j]:=A[i,2*j];
                                                  B[2,j]:=i; B[3,j]:=2*j;
                                                  sum:=sum + B[1,j];
                                                  kol:=kol + 1; jn:=j; goto MET;
```

```
                                                    end;
        B[1,j]:=0; B[2,j]:=0; B[3,j]:=0;
                          MET:end;
min:=B[1,jn]; max:=B[1,jn]; imin:=1; imax:=1; jmin:=jn; jmax:=jn;
For j:=1 to n div(2) do if B[2,j]>0 then if B[1,j]>max then
                                    begin max:=B[1,j]; imax:=round(B[2,j]);
                                    jmax:=round(B[3,j]); end
                                      else if B[1,j]<min then
                                      begin min:=B[1,j]; imin:=round(B[2,j]);
                                              jmin:=round(B[3,j]); end;
If imin<>imax then for j:=1 to n do begin
                    R:=A[imin,j]; A[imin,j]:=A[imax,j]; A[imax,j]:=R;
                                    end;
sred :=sum/kol;
end;
{-------------- Объявление объектов-переменных ------------}
Var    S : obj := new obj;   {статический объект-предок}
       SP : ob1 := new ob1;   {статический объект-предок}

{-------------- Основная программа -----------------------}
Begin  {clrscr;}
Writeln('Статические методы');
Writeln('Статический объект-предок');
{Вызовы методов для обработки полей переменной S}
 S.vvod('LR5.DAT');
 S.P;
 S.obr;
 S.vyvod;
 S.P;

Writeln('Статический объект-потомок');
{Вызовы методов для обработки полей переменной SP}
 SP.vvod('LR55.TXT');
 SP.P;
 SP.obr;
 SP.vyvod;
 SP.P;
 writeln ('среднее значение ', SP.sred);
End.
```

Результат работы программы:

```
Статические методы
Статический объект-предок

        3000   -100.40       540.8    768.10         200      -234.0
      645300   -240.07      26.765     54.90        -750     -4521.7
     -160000   -422.00      3987.5      1.00      133000     -6974.2
         899   1025.50      900.89   -400.20      270000   -111111.0
        2670    931.50      2078.9  -3229.00       93000     -2459.1
 Сформированный массив :
 1025.5  768.1    0.0
      4      1      0
      2      4      0


 Полученные результаты :

 sum = 1793.60   kol =  2

 min = 768.10    max = 1025.50

 imin = 1       imax = 4

 jmin = 2       jmax = 2
```

```
         899   1025.50      900.89   -400.20      270000   -111111.0
      645300   -240.07      26.765     54.90        -750     -4521.7
     -160000   -422.00      3987.5      1.00      133000     -6974.2
        3000   -100.40       540.8    768.10         200      -234.0
        2670    931.50      2078.9  -3229.00       93000     -2459.1
Статический объект-потомок

        1000   -700.40       140.8      8.10         100        -2.0
      645300   -240.07      26.765     54.90        -750     -4521.7
     -160000   -422.00      3987.5      1.00      133000     -6974.2
         899   1025.50      900.89   -400.20      270000   -111111.0
        2670    931.50      2078.9  -3229.00       93000     -2459.1
 Сформированный массив :
 1025.5    8.1    0.0
      4      1      0
      2      4      0


 Полученные результаты :

 sum = 1033.60   kol =  2

 min =   8.10    max = 1025.50

 imin = 1        imax = 4

 jmin = 2        jmax = 2
         899   1025.50      900.89   -400.20      270000   -111111.0
      645300   -240.07      26.765     54.90        -750     -4521.7
     -160000   -422.00      3987.5      1.00      133000     -6974.2
        1000   -700.40       140.8      8.10         100        -2.0
        2670    931.50      2078.9  -3229.00       93000     -2459.1
среднее значение 516.8
```

Пример выполнения задания 2 рассмотрим для программы, выдающей данные погоды по заданному месяцу.

```
program Z2_OOP;
type
Prognoz=class
        mes:string;
        srt:integer;
        sro:integer;
        srd:integer;
      //  procedure Create(m:string;t,o,d:integer);
    constructor Create(m:string;t,o,d:integer);
//procedure Create(m:string;t,o,d:integer);
                  begin
                     mes:=m;
                     srt:=t;
                     sro:=o;
                     srd:=d;
                  end;

     procedure Print;
        end;

procedure Prognoz.Print;
begin
writeln(mes:12,srt:4,sro:5,srd:4);
end;

const n=5;
var p:array[1..n] of Prognoz;
```

```
     m:string;
    i,t,o,d:integer;
begin
for i:=1 to n do
 begin

  writeln('Введите данные по месяцу ',i,':');
  write('- название месяца: '); readln; readln(m);
  write('- средняя температура, градусов: ');read(t);
  write('- среднее количество осадков, миллиметров: ');read(o);
  write('- среднее давление, мм. ртутного столба: ');read(d);
 //  p[i].Create(m,t,o,d);//присваиваем значения его полям
  p[i]:=Prognoz.Create ( m,t,o,d );//создаем экземпляр класса
 end;
writeln('Введенные сведения:');
for i:=1 to n do
p[i].Print;
end.
```

Фрагмент результатов представлен ниже:

```
Окно вывода
Введите данные по месяцу 1:
- название месяца: январь
- средняя температура, градусов: -20
- среднее количество осадков, миллиметров: 700
- среднее давление, мм. ртутного столба: 768
Введите данные по месяцу 2:
```

5.6. Варианты на выполнение работы №12

Вариант задания приведен кратко. Для более полного его восприятия можно воспользоваться «Типовым вариантом». Для основного задания виртуальный метод заменяется статическим.

Названия классов и функций, данные в кавычках на русском языке, заменяются в программе англоязычными, выбранными по собственному желанию.

Типовой вариант:

Создать класс "Функция", содержащий виртуальный метод "значение", принимающий один аргумент типа real и возвращающий значение того же типа.

Создать классы "Линия", "Парабола" и "Синусоида" порожденные от класса "Функция", содержащие поля для следующих параметров функций:

- "Линия" - (k,b)
- "Парабола" - (a,b,c)
- "Синусоида" - (A,w,phi)

и конструкторы, позволяющие эти параметры задавать. Каждый класс должен содержать виртуальный метод "значение", вычисляющий значение в заданной точке.

Создать процедуру "просмотр", принимающую один параметр - указатель на класс "функция". Процедура должна осуществлять вывод значений функции в задаваемых пользователем точках.

В основной программе требуется объявить переменную типа указатель на класс "функция". Запросить у пользователя тип функции (с помощью меню) и ее параметры.

Создать динамически объект заданного типа с заданными параметрами и присвоить его адрес предварительно объявленной переменной. Вызвать функцию "просмотр" и передать ей значение переменной. Удалить объект.

Варианты:

№1

Базовый класс - "Фигура", виртуальный метод - "периметр" (без параметров, возвращает real). Производные классы - "Эллипс", "Квадрат", "Треугольник".

Параметры:

- Эллипс: большая и малая оси (real)
- Квадрат: длина стороны (real)
- Треугольник: длины сторон (массив из 3-х элементов типа real)

Процедура "просмотр" выводит длину периметра переданной ей фигуры.

№2

Базовый класс - "Фигура", виртуальный метод - "объем" (без параметров, возвращает real). Производные классы - "Шар", "Призма", "Цилиндр".

Параметры:

- Шар: радиус
- Призма: длина, ширина, высота
- Цилиндр: высота, радиус основания все параметры - типа real.

Процедура "просмотр" выводит значение объема переданной ей фигуры.

№3

Базовый класс - "Числовой ряд", виртуальный метод - "значение члена ряда" (параметр - номер члена ряда (longint), возвращает real).

Базовый класс содержит невиртуальный метод вычисления частичной суммы ряда (параметр - число суммируемых элементов, возвращает real).

Производные классы - "Гармонический ряд" (1/n), "Ряд экспоненты" (1/n!)

Процедура "просмотр" выводит значение частичной суммы переданного ей ряда (число членов частичной суммы задается пользователем).

№4

Базовый класс - "Функциональный ряд", виртуальный метод - "значение функции-элемента ряда в заданной точке" (параметры - номер элемента (longint) и значение аргумента функции(real), возвращает real).

Производные классы:

- "Ряд синуса" {((-1)^n)*(x^(2n+1))/(2n+1)!},
- "Ряд косинуса" {1 при n=0, ((-1)^n)*(x^2n)/(2n)! при n>0}

Процедура "просмотр" выводит значения выбранной пользователем функции-элемента ряда при выбранных пользователем значениях аргумента.

№5

Базовый класс - "Число", виртуальный метод - "модуль" (без параметров, возвращает real). Производные классы - "Действительное число", "Комплексное число".

Параметры:

- Действительное число: значение
- Комплексное число: значения действительной и мнимой частей все параметры - типа real.

Процедура "просмотр" выводит значение модуля переданного ей число.

№6

Базовый класс - "Электронагревательный прибор", виртуальный метод – "выделяемое тепло" (в единицу времени) (без параметров, возвращает real).

Производные классы - "Кипятильник", "Электроплитка", "Электрообогреватель".

Параметры:

- Кипятильник: количество витков, радиус витка, мощность теплоизлучения, выделяемая единицей длины витка
- Электроплитка: площадь и мощность теплоизлучения, выделяемая единицей площади
- Электрообогреватель: количество пластин, площадь пластины с одной стороны и мощность теплоизлучения, выделяемая единицей площади

Все параметры - типа real.

Вычисление мощности теплоизлучения:

Кипятильник: количество витков*2*pi*радиус витка*мощность теплоизлучения выделяемую единицей длины витка.

Электроплитка: площадь*мощность теплоизлучения выделяемую единицей площади.

Электрообогреватель: количество пластин * площадь пластины с одной стороны *2* мощность теплоизлучения, выделяемую единицей площади.

Процедура "просмотр" выводит значение выделяемого тепла переданного ей электронагревательного прибора.

№7

Базовый класс - "Множество чисел", виртуальный метод - "принадлежность числа множеству" (параметр - число (real), возвращает boolean). Производные классы - "Отрезок", "Пересечение", "Объединение".

Параметры:

- Отрезок: начало и конец отрезка (оба real)
- Пересечение: указатели на два пересекаемых множества
- Объединение: указатели на два объединяемых множества

Процедура "просмотр" определяет и выводит, принадлежат ли заданные пользователем числа переданному ей множеству.

№8

Базовый класс - "Печатное издание", виртуальный метод - "стоимость" (без параметров, возвращает real). Производные классы - "Книга", "Газета", "Журнал"

Параметры:

- Книга: количество страниц, тип переплета (жесткий, мягкий), наличие иллюстраций
- Газета: количество страниц, цветность (цв., ч/б)
- Журнал: количество страниц, тип бумаги (матовая, глянцевая)

Вычисление стоимости:

Книга: K1*количество страниц+K2*M+K3*N

- M=в зависимости от типа переплета: жесткий - 3, мягкий - 1
- N=при наличии иллюстраций - 1, при отсутствии - 0

Газета: K4*количество страниц*M

- M=при цветной бумаге - 1.5, при черно-белой - 1

Журнал: K5*количество страниц*M

- M=в зависимости от типа бумаги: матовая - 1, глянцевая - 2.5

Константы стоимости K1-K5 - на собственное усмотрение. Процедура "просмотр" выводит стоимость переданного ей печатного изделия.

№9

Базовый класс - "Математическое выражение", виртуальный метод - "значение" (без параметров, возвращает real). Производные классы - "Константа", "Сумма", "Произведение".

Параметры:

- Константа: "значение" (real)
- Сумма: указатели на два суммируемых выражения
- Произведение: указатели на два перемножаемых выражения

В основной программе: создать переменную для выражения "ax^2+b", выводить значения выражения для задаваемых пользователем значений x.

№10

Базовый класс - "Выкройка", виртуальный метод - "требуемое количество материала" (без параметров, возвращает real). Производные классы: "Брюки", "Пальто", "Шапка"

Параметры:

- Брюки: Количество штанин (byte), длина (real), полуокружность бедер (real)
- Пальто: длина (real), полуокружность груди (real), длина рукава (real)
- Шапка: радиус головы (real)

Вычисление количества материала:

- Брюки: Количество штанин*длина*(полуокружность бедер/2+6)
- Пальто: длина*(полуокружность груди+10)+длина рукава*(полуокружность груди/3+8)
- Шапка: 2*pi*(радиус головы)^2

Процедура "просмотр" выводит требуемое количество материала для пошива переданного ей изделия.

№11

Базовый класс - "Конденсатор", виртуальный метод - "емкость" (без параметров, возвращает real). Производные классы: "Плоский конденсатор", "Шаровой конденсатор".

Параметры:

- Плоский конденсатор: площадь поверхности пластин, расстояние между пластинами
- Шаровой конденсатор: радиусы внутренней и внешней сферы

Процедура "просмотр" выводит емкость переданного ей конденсатора.

№12

Базовый класс - " Функция ", виртуальный метод - " значение " (принимает и возвращает real). Производные классы - "Прямая", "Кубическая парабола" и "Гипербола".

Параметры:

- " Прямая " - (k,b)
- " Кубическая парабола " – без параметров
- " Гипербола " - (k)

Процедура "просмотр" должна осуществлять вывод значений функции в задаваемых пользователем точках.

№13

Базовый класс - "Итоговая функция", виртуальный метод - "значение" (без параметров, возвращает real). Производные классы - "Максимум", "Минимум", "Среднее".

Параметры:

- Максимум: "значение" (integer)
- Минимум: указатели на два суммируемых выражения (integer, k)
- Среднее: указатели на два перемножаемых выражение (real, k1, k2)

В основной программе создать (ввести) целочисленный массив, выводить значения:

- Максимума из всех элементов массива;
- Минимума из первых k элементов массива;
- Среднего значения элементов, начиная с k1до k2.

№14

Базовый класс - "Фигура", виртуальный метод - "площадь" (без параметров, возвращает real). Производные классы - "Окружность", "Прямоугольник", "Треугольник".

Параметры:

- Окружность: радиус (real)
- Прямоугольник: ширина (real), высота (real)
- Треугольник: длины сторон(массив из 3-х элементов типа real)

Процедура "просмотр" выводит значение площади переданной ей фигуры.

№15

Базовый класс - " Функция ", виртуальный метод - " значение " (принимает и возвращает real). Производные классы - "Линия", "Парабола" и "Синусоида".

Параметры:

- "Линия" - (k,b)
- "Парабола" - (a,b,c)
- "Синусоида" - (A,w,phi)

Процедура "просмотр" должна осуществлять вывод значений функции в задаваемых пользователем точках.

№16

Базовый класс - "Жилье", виртуальный метод - "стоимость" (без параметров, возвращает real). Производные классы - "Квартира", "Дача", "Трейлер" (дом на колесах).

Параметры:

- Квартира: метраж (real), местоположение (перечисление: центр, окраина, за кольцевой)
- Дача: количество этажей (byte), размер участка (real)
- Трейлер: скорость (byte), наличие биотуалета (boolean)

Вычисление стоимости:

Квартира: K1*M*метраж

- M=в зависимости от местоположения: центр - 2, окраина - 1, за кольцевой - 0.5

Дача: K2*количество этажей+K3*размер участка

Трейлер: K4*скорость+B

- B=0 при отсутствии биотуалета, B=K5 при наличии

Константы стоимости K1-K5 - на собственное усмотрение.

Процедура "просмотр" выводит значение стоимости переданного ей "жилья".

№17

Базовый класс - "Фигура", виртуальный метод - "рисовать" (без параметров, без возвращающего значения). Производные классы - "Окружность", "Прямоугольник", "Линия"

Параметры:

- Окружность: (x,y,r)
- Прямоугольник: (x1,y1,x2,y2)
- Линия: (x1,y1,x2,y2)

Все параметры - типа integer.

Рисование - в графическом режиме. Процедура "просмотр" рисует объект на экране.

№18

Базовый класс - "Вещь", виртуальный метод - "информация" (без параметров, без возвращаемого значения). Производные классы - "Книга", "Яблоко", "Купюра"

Параметры:

- Книга: название (string), автор (string), количество страниц (longint)
- Яблоко: вес (float), цвет (перечисление)
- Купюра: достоинство (longint)

Метод "информация" должен вывести информацию об объекте в текстовом виде на экран. Информация включает в себя тип объекта и значение его параметров.

Процедура "просмотр" выводит информацию об объекте.

№19

Базовый класс - "Фауна", виртуальный метод - "информация" (без параметров, без возвращаемого значения). Производные классы - "Млекопитающее", "Рыба", "Птица"

Параметры:

- Млекопитающее: название (string), класс (string), среда обитания (string), средняя продолжительность жизни (float)
- Рыба: название (string), тип (string), средний вес (float)
- Птица: название (string), размах крыльев (longint)

Метод "информация" должен вывести информацию об объекте в текстовом виде на экран. Информация включает в себя тип объекта и значение его параметров.

Процедура "просмотр" выводит информацию об объекте.

№20

Базовый класс - "Машина", виртуальный метод - "информация" (без параметров, без возвращаемого значения). Производные классы - "Грузовик", "Легковая", "Автобус"

Параметры:

- Грузовик: марка (string), число цилиндров (integer), грузоподъемность (longint)
- Легковая: марка (string), модель (string), число цилиндров (integer)
- Автобус: марка (string), количество пассажиров (integer)

Метод "информация" должен вывести информацию об объекте в текстовом виде на экран. Информация включает в себя тип объекта и значение его параметров.

Процедура "просмотр" выводит информацию об объекте.

№21

Построить три класса (базовый и 3 потомка), описывающих некоторых хищных животных (один из потомков), всеядных (второй потомок) и травоядных (третий потомок).

Параметры: название, класс, вид, кличка, возраст, вес, тип пищи.

Описать в базовом классе абстрактный метод для расчета количества и типа пищи, необходимого для пропитания животного в зоопарке.

- Вывести информацию о каждой группе животных.
- Упорядочить всю последовательность животных по убыванию количества пищи. При совпадении значений – упорядочивать данные по алфавиту по имени. Вывести идентификатор животного, имя, тип и количество потребляемой пищи для всех элементов списка.
- Вывести последние 5 названий и имен животных из полученного в предыдущем пункте списка.

№22

Базовый класс - "Счетчик", виртуальный метод - "текущее состояние" (без параметров, без возвращаемого значения). Производные классы - "Десятичный счетчик", "Двоичный счетчик", "Символьный счетчик"

Метод "информация" должен вывести информацию об объекте на экран. Информация включает в себя тип счетчика и его текущее значение.

Счетчик может увеличивать или уменьшать свое значение на единицу в заданном диапазоне. Предусмотреть инициализацию счетчика значениями по умолчанию и произвольными значениями. Счетчик имеет два метода: увеличения и уменьшения.

5.7. Контрольные вопросы

1. Что такое объект?
2. Как объявить объект на Паскале?
3. Как объявить динамический объект на Паскале?
4. Как вызвать метод для статического объекта?
5. Что такое объект-предок и объект-потомок?
6. Для каких целей используют объект-потомок?
7. Какие свойства лежат в основе объектно-ориентированного программирования и как они реализованы в Паскале?
8. Какого типа могут быть поля объекта?
9. Что такое виртуальный метод?
10. Когда и для каких целей используется конструктор?
11. Когда и для каких целей используется деструктор?

6. Лабораторная работа №13

Разработка программы обработки графов

6.1. Цель работы

Целью работы является освоение

- способов представления графов;
- основных задач, решаемых с помощью графов;
- основных алгоритмов обработки графов и процедур обработки графовых моделей.

6.2. Задание на выполнение работы

Разработать программу, выполняющую обработку графов в соответствии с вариантом. При этом:

- Отображать вводимые исходные данные;
- При необходимости обеспечивать интерактивную обработку (диалог с пользователем), например, для выбора вершины.

6.3. Требования к выполнению

- Использовать треугольные и диагональные массивы, если матрицы симметричны;
- Результат снабжать комментариями и пояснительными надписями.

6.4. Краткие теоретические сведения

6.4.1 Массивы

Массивы – наиболее часто используемые структуры данных и во многих случаях самые удобные. Но при этом далеко не всегда это оптимально с точки зрения экономии ресурсов ПК. Так, например, реализация специальных типов массивов нестандартными средствами может привести к значительной экономии оперативной памяти. Об этом и пойдет речь в данном разделе. Кстати, некоторые системы программирования имеют специальные подпрограммы для работы с такими массивами.

Треугольные массивы.

Предположим, у нас есть информация об N городах и расстояниях между ними. При помощи массива можно построить матрицу смежности, хранящую информацию о расстоянии между городами, но совершенно очевидно, что на диагонали элементы не будут иметь значений, а остальная информация будет дублирована, т.к. A[I,J] = A [J,I]. Такие массивы и называют треугольными.

При больших значениях N излишние затраты памяти могут быть существенны. Для N городов будет N*(N-1) дублированных элементов и N подобных A[I,I]. Если N=1000, то в массиве будет храниться более 500 тысяч ненужных элементов.

Избежать таких потерь можно, создав одномерный массив и упаковав в него двумерный.

Формула для преобразования A[I,J] в B[X] имеет следующий вид:

X := round (I*(I-1)/2)+J для I > J.

В данном случае нумерация элементов массивов начинается с 0. В этом случае все используемые формулы получаются более простыми.

Диагональные массивы.

В некоторых случаях используются треугольные массивы, использующие диагональные элементы. Такие массивы и называются диагональными. В этом случае требуется сделать два изменения в формуле преобразования индексов:

Преобразование не должно отклонять случаи I = J,

Перед вычислением индекса в массиве B необходимо добавить к I единицу.

Нерегулярные массивы.

В некоторых программах требуются массивы с нестандартной формой. Например, в каждой строке двумерного массива нужно хранить разное количество элементов (например, координаты вершин многоугольников, каждый из которых имеет разное количество вершин).

Можно было бы выровнять такой массив по большему количеству элементов, но в этом случае возможны потери памяти, необходимость хранения количества элементов в каждой строке обуславливает использование второй сложной структуры, а также разработку системы их связывания.

Нерегулярные многомерные массивы организуют с помощью двух массивов меньшей размерности либо массива и связанного списка.

Например, двумерный массив можно представить как два одномерных: один содержит все элементы двумерного, в котором элементы строк размещаются последовательно друг за другом, а второй – номера элементов в первом массиве, где начинается каждая строка. Для удобства обработки во второй массив можно поместить последний элемент, который укажет на элемент, следующий за последним.

Второй случай аналогичен, но вместо номера первого элемента каждой строки нужно хранить во втором массиве адреса в списке первых элементов строк.

Разреженные массивы.

Иногда программы используют большие массивы, которые содержат много нулевых элементов. Такие массивы называют разреженными. При более 50% ненулевых элементов массивы также называют сильно разреженными.

Для организации многомерных разреженных массивов используется одномерный массив, содержащий значения указателей на списки, где элементами списков являются элементы каждой строки массива или, при большей исходной размерности, сами являются указателями на списки. Последнее ввиду сложности обработки используется редко. Разреженные массивы размерности больше двух практически в реальном программировании не встречаются.

В случае сильно разреженных массивов предполагается, что массив вообще может содержать пустые строки, и в этом случае его обработка через массив списков не оптимальна. Массив можно заменить списком. В таком случае будет обработка списка списков. На эту обработку распространяются все правила рассмотренные ранее в разделе о линейных связных структурах. Основное отличие – пересчет индексов в порядковые номера в списках.

6.4.2. Граф

Граф состоит из множества вершин и множества дуг, вершины также называют узлами, а дуги ребрами. Дуга представляется в виде пары вершин.

Вершины графа можно использовать для представления объектов, а дуги для отношений между объектами (например, вершины – города, а дуги – дороги между городами).

Если пары узлов упорядочены, то граф называют направленным или ориентированным. Если с дугами графа связано какое-либо значение, то граф называют взвешенным или помеченным. Если граф содержит циклы, то он называется циклическим.

Примеры ориентированного и неориентированного графа представлены на рисунке 1.

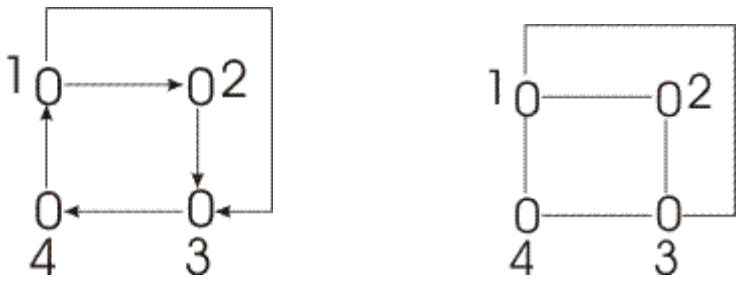


Рисунок 1. Ориентированный и неориентированный графы

Для представления графов можно использовать различные структуры данных. Выбор структуры зависит от операторов, которые будут применяться к вершинам и дугам графа. Одним из наиболее общих представлений является ***матрица смежности***.

Матрица смежностей неориентированного графа - это матрица B=[Bij] размерности |V|x|V|, где

$$B_{i,j} = \begin{cases} 1, \text{если существует ребро, идущее из вершины i в вершину j;} \\ 0, \text{в противном случае.} \end{cases}$$

Матрица смежностей ориентированного графа - это матрица B=[Bij] размерности |V|x|V|, где

$$B_{i,j} = \begin{cases} 1, \text{если существует дуга, идущая из вершины i в вершину j;} \\ 0, \text{в противном случае.} \end{cases}$$

Легко видеть, что матрица смежностей неориентированного графа всегда симметрична, поэтому для ее представления достаточно хранить только ее верхний треугольник.

Матрицы смежности для графов, представленных на рисунке 1, представлены на рисунке 2:

$$\begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix} \qquad \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

Рисунок 2. Матрицы смежности неориентированного и ориентированного графов.

В качестве единиц и нулей могут использоваться как арифметические константы, так и логические.

Для помеченных графов матрица смежности будет заполняться не единицами, а метками (т.е. соответствующими значениями дуг). Вместо нулей будут использоваться специальные символы, означающие, что клетка пуста. Ноль в данном случае допустим также, но только в том случае, если он ни при каких условиях не может оказаться значением, приписанным дуге (ребру).

Представление графа с помощью матрицы смежностей зачастую неудобно, поскольку количество вершин требуется знать заранее. Если граф должен создаваться или изменяться во время исполнения программы, то для каждого добавления или удаления вершины необходимо строить новую матрицу. Кроме того, даже если граф содержит малое число ребер (дуг) и матрица смежностей состоит в основном из нулей, память должна быть отведена для всех возможных дуг вне зависимости от того, существуют ли они. Если граф содержит n вершин, то должна быть отведена память для n2 элементов. Как и следовало ожидать, возможное решение - это использовать динамическую (связанную) структуру данных для представления графа. В этом случае будет более удобным применение классической структуры, называемой **списком смежности.**

Список смежности содержит для каждой вершины графа список смежных ей вершин. Используя терминологию языка Pascal, можно утверждать, что каждый элемент такого списка является записью R, содержащей в поле R.Stroka вершину графа, а в поле R.Next - указатель на следующую запись в списке (ясно, что для последней записи в списке R.Next содержит Nil).

Обозначим Beg[v] - указатель на начало списка, содержащего вершины, смежные с вершиной v. Изобразим схематически представление неориентированного графа с помощью списков смежности (отметим, что для неориентированных графов каждое ребро представлено в списках смежности дважды).

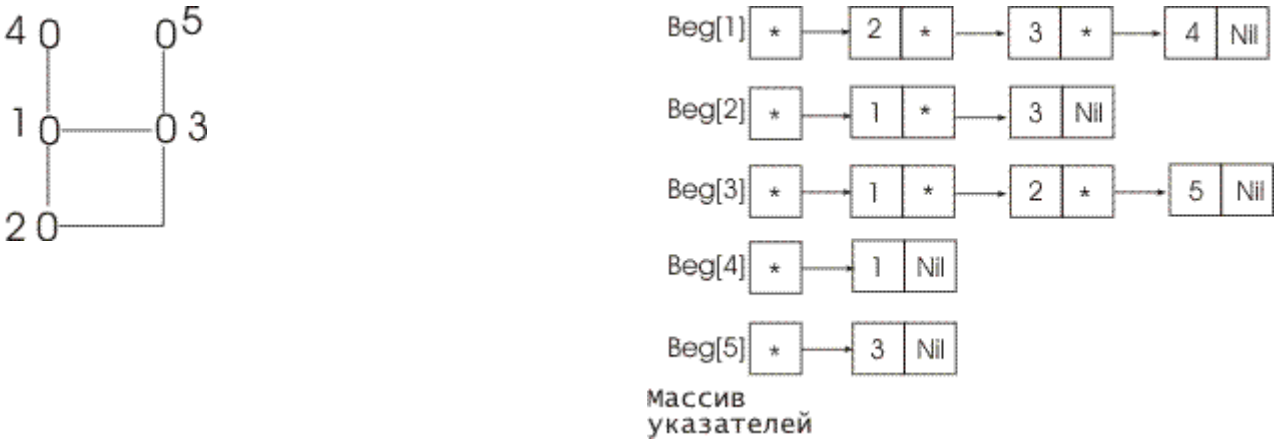


Рисунок 3. Граф и его список смежности

На примере, представленном на рисунке 3, структура Beg – это одномерный массив указателей на списки. Но в том случае, если в процессе работы будут меняться не только связи между узлами, но и количество узлов, такую структуру более удобно представлять в виде списка.

К графу можно применить следующие операции:

- Вставка вершин и дуг (ребер).
- Удаление вершин и дуг.
- Чтение меток вершин и дуг.
- Просмотр вершин.
- Удаление графа.

Все операции, кроме просмотра, определяются исключительно типом структуры данных, используемой для реализации графа, поэтому в этом разделе остановимся только на одной операции – просмотр вершин или оператор перемещения по последовательности дуг.

Рассмотрим его подробнее. Для реализации этой операции необходим индексный тип структуры данных, но сам индекс может пониматься по-разному в зависимости от структуры. В основе большинства операций для работы с графами лежит оператор:

For каждая вершина W, смежная с вершиной V do

{некоторые действия с вершиной W}.

Для просмотра множества вершин необходимы три оператора, возвращающие:

1. Индекс первой вершины, смежной с заданной вершиной (если вершина не имеет смежных, то возвращается «пустая» вершина). Назовем ее First (V).
2. Индекс вершины, смежный с заданной вершиной, следующий за индексом I (если вершина I последняя в списке, то возвращается «пустая» вершина). Назовем ее Next (V,i).
3. Вершину с индексом I из множества вершин, смежных с заданной. Назовем ее VerCur (V).

Во всех случаях под заданной вершиной понимаем вершину V.

Рассмотрим эти операции при условии, что граф представлен матрицей смежности A размером N x N.

```
Function First (V: integer) : integer;
      Var I : integer;
Begin
       For I := 1 to N do
                If A [V,I] then return (I);
       Return (0);
End;
```

```
Function Next (V: integer, i: integer) : integer;
      Var J : integer;
Begin
       For J := I + 1 to N do
                If A [V,J] then return (J);
       Return (0);
End;
```

Тогда последовательный просмотр вершин, смежных с вершиной V можно записать в следующем виде:

```
I:= First ( V );
While I <> 0 do begin
           W:= VerCur (V,i);    //действия с вершиной W
           I:= Next (V, i);
End;
```

6.5. Пример выполнения работы

В данной работе типовых заданий нет, поэтому в этом разделе приводятся две тривиальные задачи, которые могут быть использованы как часть лабораторной работы.
Задача 1. Сформировать матрицу транзитивного замыкания, используя алгоритм Уолша.

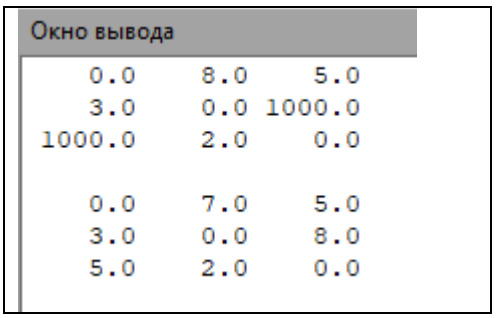
```
program graf; //Warshell
Const N = 3;
type Tm = array [1..N, 1..N] of byte;
Var A, C : Tm; i,j : byte;

procedure Vivod(A: Tm);
Var i,j : byte;
begin
For i:= 1 to N do begin
     For j := 1 to N do write (A [i,j]: 3);
     writeln; end;
end;

// алгоритм Warshell
Procedure Warshell ( var A: Tm; C : Tm);
Var i,j,k : byte;
begin
For i:= 1 to N do
     For j := 1 to N do A [i,j] := C [i,j];
For k:= 1 to N do
     For i:= 1 to N do
          For j := 1 to N do
               if A [i,j] = 0 then
                    A [i,j] := A [i,k] and A [k,j];
end;

// основная программа
begin
Assign (input, 'tr_zam.pas'); Reset (input);
For i:= 1 to N do
     For j := 1 to N do read (C [i,j]);
Vivod (C);
 writeln;
Warshell (A,C);
Vivod (A);
end.
```

Результат работы программы:

Задача 2. Сформировать матрицу расстояний, используя алгоритм Флойда.

```
program graf;
Const N = 3;
type Tm = array [1.. N, 1.. N] of real;
Var A, C : Tm; i,j : integer;

procedure Vivod(A: Tm);
Var i,j : integer;
begin
For i:= 1 to N do begin
     For j := 1 to N do write (A [i,j]: 7:1);
     writeln; end;
end;

// алгоритм Флойда
Procedure Floyd ( var A: Tm; C : Tm);
Var i,j,k : integer;
begin
For i:= 1 to N do
     For j := 1 to N do A [i,j] := C [i,j];
For i:= 1 to N do A [i,i] := 0;

For k:= 1 to N do
     For i:= 1 to N do
          For j := 1 to N do
               if A [i,k] + A [k,j] < A [i,j] then
                    A [i,j] := A [i,k] + A [k,j];
end;

// основная программа
begin
Assign (input, 'floyd.pas'); Reset (input);
For i:= 1 to N do
     For j := 1 to N do read (C [i,j]);
Vivod (C);
 writeln;
Floyd (A,C);
Vivod (A);
end.
```

Результат работы программы (здесь 1000 – «большое» значение):



6.6. Варианты на выполнение работы №13

1. Между 9 планетами Солнечной системы введено космическое сообщение. Ракеты летают по следующим маршрутам: Земля–Меркурий, Плутон–Венера, Земля–Плутон, Плутон–Меркурий, Меркурий–Венера, Уран–Нептун, Нептун–Сатурн, Сатурн–Юпитер, Юпитер–Марс и Марс–Уран. Определите, можно ли добраться с Земли до заданной планеты.
2. Дана матрица смежности. Получить список четных вершин, имеющих связь только с нечетными вершинами.
3. Дан граф (сильно разреженная матрица), содержащий расстояния между небольшими населенными пунктами (дороги). Определите населенный пункт, где должна быть размещена школа, если известно количество жителей этого населенного пункта и процент детей.
4. Дано множество задач по проекту T_1, T_2, …, T_n, для выполнения которых необходимо время t_1, t_2, …, t_n, и множество отношений вида «задача T_i должна закончиться до начала задачи T_j». Найдите минимальное время для выполнения всех задач.
5. В соревнованиях по борьбе, проходящих по олимпийской системе, участвуют N борцов. Определить минимальное время, за которое можно провести соревнование, если в спортивном зале есть только K борцовских ковров, и на каждую схватку, включая разминку и отдых, отводится час.
6. Дан граф (сильно разреженная матрица), содержащий расстояния между небольшими населенными пунктами (дороги). Определите населенный пункт, где должна быть размещена пожарная каланча (Возможно размещение на дороге).
7. Дан граф, содержащий расстояния между городами. Реализовать программу обхода городов таким образом, чтобы суммарное расстояние было наименьшим, но ни в один город нельзя заходить дважды.
8. Дана матрица смежности. Определить степень всех вершин графа.
9. Дан граф (сильно разреженная матрица), содержащий расстояния между городами. Сформировать программу, которая определяет кратчайшее расстояние между любыми двумя городами.
10. В деревне есть N телефонов, а АТС отсутствует. Нужно телефоны соединить проводами так, чтобы длина проводов была минимальной, а каждый телефон соединялся не более, чем с K телефонами.
11. Дана матрица смежности. Получить список вершин, у которых число дуг больше заданного N.
12. Найти кратчайшие пути в орграфе от первой вершины ко всем остальным, используя алгоритм Дейкстры. Постройте дерево кратчайших путей.

13. Дана матрица смежности. Постройте остовное дерево минимального веса, используя алгоритмы Прима и Краскала.
14. Дан граф (сильно разреженная матрица), содержащий расстояния между небольшими населенными пунктами (дороги). Определите населенный пункт, где должна быть размещена пожарная каланча (В населенном пункте без учета численности его населения).
15. Дана матрица смежности. Определить все пути между двумя заданными городами.
16. Дана матрица смежности. Получить список вершин, у которых число входов больше, чем число выходов.
17. Дана матрица смежности. Найдите количество остовных деревьев и выведите их.
18. Дана матрица смежности. В узлах содержаться целые числа. Напишите программу обхода графа в глубину и в ширину. В процессе обхода определять минимальное и максимальное значения узлов.
19. Дан граф. Написать программу алгоритма Дийкстры, используя связанные списки смежности (реализованную посредством частично упорядоченного дерева).
20. Дан граф (сильно разреженная матрица), содержащий расстояния между небольшими населенными пунктами (дороги). Определите населенный пункт, где должна быть размещена школа, если известно количество жителей этого населенного пункта и процент детей.

6.7. Контрольные вопросы

1. Как объявить граф?
2. Что такое матрица смежности и как ее задать?
3. Поясните алгоритм Дийкстры.
4. Поясните алгоритмы Прима и Краскала.
5. Что такое список смежности?
6. Что такое треугольные и диагональные массивы?
7. Как задавать разреженные массивы?
8. Какие операции можно применить к графу?
9. Поясните алгоритм Уолша.
10. Что такое оргграф?
11. Что такое остовное дерево и как его найти?

7. Общие подходы к решению задач

Рекомендации и общие подходы к решению задач были обнаружены в книге А.Спрола «Думай как программист: креативный подход к созданию кода» («ЭКСМО», 2018, 272 с.). Книга написана для языка C++, однако именно этот аспект является независимым от языка и крайне полезным, особенно для начинающего программиста.

1. Планируйте решение.

Даже если задача небольшая не начинайте немедленно программу писать. Продумайте ход решения, структуру, способ и форму ввода данных (инициализации), результат работы программы. Продумайте промежуточные этапы контроля программы (установите промежуточные цели). По ходу решения план может измениться, это не страшно. «Хаос», который получается при программировании необдуманной задачи, гораздо хуже.

2. Перефразируйте задачи.

Это необходимо для уточнения условия задачи, обдумывания решения, проектирования решения и т.п. Возможно, при перефразировании окажется, что задача проще, чем вы думаете (или сложнее). Или есть неточности или неясности в условии. Как показала практика, если студент не может перефразировать задачу (или просто повторить ее, никуда не глядя), то значит он ее не понимает и к решению не готов.

3. Делите задачи.

Разделение задачи на несколько частей (шагов/этапов) может значительно упростить задачу, особенно процесс отладки. К тому же минимизация при программировании вложенных операторов упрощает не только разработку, но и чтение программы, а часто приводит и к снижению процессорной нагрузки.

4. Начинайте с того, что знаете.

Как бы не была сложна задача, разделив ее на части можно получить подзадачи, с решением которых уже сталкивались или найти их решение несложно. После этого можно понемногу встраивать в программу новые фрагменты.

5. Упрощайте задачи.

Полезно использовать прототипы - чуть менее сложные решения, чем это необходимо для решения поставленной задачи.

6. Ищите аналогии.

Возможно, вы такую задачу уже решали. Или она похожа на решенную ранее задачу. Аналогии не всегда прямолинейны, но научиться распознавать аналогии крайне важно.

7. Экспериментируйте.

Иногда полезно для прогресса что-то попробовать сделать и понаблюдать за результатами. Но это должен быть контролируемый процесс с ожидаемым результатом, который будет или не будет подтвержден.

8. Не расстраивайтесь.

И не раздражайтесь. Программа не напишется и не отладится сама. А вам потребуется «холодная голова», иначе четко мыслить вы не будете. Вернитесь на шаг назад (или на два, где программа еще работала) и начните спокойно снова.

8. Список рекомендуемой литературы

1. Егорова А.А.	Пособие по проведению практических занятий по дисциплине «Алгоритмические языки и программирование» - М.: МГТУ ГА, 2013 – 36 стр.
2. Егорова А.А.	Алгоритмические языки и программирование. Учебно-методическое пособие по выполнению лабораторных работ. Часть 1. Основы программирования. - М.: МГТУ ГА, 2020 – 52 стр.
3. Егорова А.А.	Алгоритмические языки и программирование. Учебно-методическое пособие по выполнению лабораторных работ. Часть 2. Модульное программирование. - М.: МГТУ ГА, 2021 – 52 стр.
4. Климова Л.М.	Pascal 7.0. Практическое программирование. Решение типовых задач. КУДИЦ-ОБРАЗ., 4-е издание
5. Кремень Ю., Расолько Г.	Теория и практика программирования на языке Pascal / Москва / Литрес/ 2015 г., 449 с.
6. Осипов А.В.	PascalABC.NET: Введение в современное программирование.–

	Ростов-на-Дону , 2019 – 572 с. :
7.	https://pascal-abc.ru.net/uchebnik/
8.	http://pascalabc.net/downloads/OsipovBook/%D0%9A%D0%BD%D0%B8%D0%B3%D0%B0%D0%94%D0%BB%D1%8F%D0%A1%D0%B0%D0%B9%D1%82%D0%B0.pdf
9.	http://pascalabc.net/

9. Заключение

Поскольку лабораторные работы, а соответственно и данное учебное пособие имеют практическую направленность и предназначены в первую очередь для отработки навыков, то и защита каждой лабораторной работы проводится с демонстрацией полученных навыков. А значит, их необходимо довести если не до автоматизма, то хотя бы до такого уровня, при котором можно выполнить задание преподавателя за ограниченное время, не гладя в пособие. Это требует определенной практики. Поэтому выполнение каждой работы – это не просто четкое выполнение задания, но и практическое изучение материала, связанного с заданием. Вариативность выполнения каждого задания (несколькими способами) позволит лучше понять механизм функционирования систем и закрепить навык.

В рамках лабораторной работы приветствуется расширение работы (использование не указанных в пособии функций, операций, выполнение дополнительных заданий и т.п.).