УДК [004.421+004.438](076.5)

МЕТОДИКА БЫСТРОГО ОБУЧЕНИЯ ПРОГРАММИРОВАНИЮ НА ОСНОВЕ ИЗУЧЕНИЯ КЛАССОВ ЗАДАЧ (18–19)

***Юрий Александрович Аляев**, доц., доц. кафедры программного обеспечения вычислительной техники и автоматизированных систем,*
e-mail: alyr1@yandex.ru,
Пермский военный институт внутренних войск МВД России,
http://pvivv.ru

Предлагается методика быстрого обучения программированию на основе изучения классов задач, разработанная и применяющаяся на практике в процессе обучения программированию студентов вузов.

Ключевые слова: алгоритм; программа; язык программирования Паскаль; массив; указатели; динамическая память; структуры данных; отладка

DOI: 10.21777/2312-5500-2016-4-3-13

Введение


Ю.А. Аляев

Разделы курса «Информатика» – алгоритмизация и программирование – остаются наиболее важными для формирования алгоритмического мышления. Поскольку в школах данные разделы преподаются в недостаточном объеме, в вузе возникает необходимость начинать обучение с нуля и достичь хорошего уровня программирования при ограниченном количестве часов преподавания.

Этого удается добиться за счет применения рациональных методов обучения, прежде всего последовательно проводя идеи обучения на основе выделения элементарных операций деятельности по построению алгоритмов и программ; выявления структуры алгоритма и форм ее записи на алгоритмическом языке; одинаковой формы алгоритма для решения задач с одинаковой структурой исходных данных [1, 2]. Благодаря этим идеям, задачи по программированию удается разбить на ряд классов и типизировать методы решения задач каждого класса.

Предлагаемая методика быстрого обучения программированию на основе изучения классов задач, появилась и применяется на протяжении многих лет в процессе обучения программированию студентов пермских вузов благодаря В. П. Гладкову [2]. В статье рассматриваются методики решения по двум (18–19) из девятнадцати выделенных классов задач (1–17 классы задач рассмотрены в [3–8]).

18. Указатели. Динамическая память. Структуры данных

Задача 1. Создать динамический одномерный массив. Заполнить его случайными целыми трехзначными числами. Найти среднее арифметическое элементов с четными и нечетными индексами. Переписать в другой динамический массив элементы (с четными или нечетными индексами), у которых среднее арифметическое больше. Распечатать элементы вновь созданного массива, которые больше его среднего арифметического.

Решение. Приводится одна из методик организации динамического массива. При объявлении типа диапазон изменения индексов задается как [1..1]. Для того чтобы не возникло ошибки при выходе за границу массива, необходимо отключить директиву компилятора {$R-}.

```
type mas = array[1..1]of integer;
var a1,                    {исходный массив}
  a2:^mas;                 {результирующий массив}
```

```
  p:pointer;      {указатель для запоминания состояния памяти}
  n,              {количество элементов в массиве <=16000}
  i:integer;      {индекс массива}
  s1,s2:real;     {среднее арифметическое для элементов с четными и нечетными
индексами соответственно}
  k1,k2:integer; {количество элементов с четными и нечетными индексами соот-
ветственно}
begin write('Введите 2<=n<=16000 ');
  readln(n);
  {$R-}          {отключили проверку выхода индекса за объявленный диапазон}
  mark(p); {запомнили состояние динамической памяти}
  getmem(a1,n*sizeof(integer)); {выделили память для массива a1}
  for i:=1 to n do a1^[i]:=100+random(900); {сформировали массив a1}
  s1:=0;s2:=0; k1:=0;k2:=0;
  for i:=1 to n do if i mod 2 =0
                then begin s1:=s1+a1^[i]; k1:=k1+1; end
                else begin s2:=s2+a1^[i]; k2:=k2+1; end;
          release(p); {освободили память до адреса, записанного в переменной p}
          s1:=s1/k1; s2:=s2/k2;
          if s1<s2
          then begin getmem(a2,k2*sizeof(integer));
                 i:=1; while i<=n do
                          begin a2^[i div 2+1]:=a1^[i];
                                  i:=i+2;
                          end;
                         for i:=1 to k2 do if a2^[i]>s2 then write(a2^[i],' ')
              freemem(a2,k2*sizeof(integer));
                    end
          else begin getmem(a2,k1*sizeof(integer));
                 i:=2; while i<=n do
                          begin a2^[i div 2]:=a1^[i];
                                  i:=i+2;
                          end;
                          for i:=1 to k1 do if a2^[i]>s1 then write(a2^[i],' ');
          freemem(a2,k1*sizeof(integer));
                    end;
end.
```

Задача 2. Создать динамический двухмерный массив, заполнить его случайными цифрами, затем распечатать.

Решение. Аналогично решению предыдущей задачи 16.2.1. получим следующую программу.

```
type item=real;
  row=array[1..1] of item;
  prow=^row;
  mas=array[1..1] of prow;
  pmas=^mas;
var r:pmas;
  m,n,                {размерности массива}
  i,j:integer;        {индексы}
begin {$R-}
  write('Введите количество строк массива '); readln(m);
  write('Введите количество столбцов массива '); readln(n);
```

```
   getmem(r,m*sizeof(prow));
   for i:=1 to m do getmem(r^[i],n*sizeof(item));
   for i:=1 to m do
   begin writeln;
         for j:=1 to n do begin r^[i]^[j]:=random(10);
         write(r^[i]^[j],' ');
    end;
   for i:=1 to n do freemem(r^[i],n*sizeof(item));
   freemem(r,m*sizeof(prow));
end.
```

Задача 3. Создать динамический массив для размещения матрицы 100 на 200 вещественных чисел. Для обращения к элементам массива написать процедуры или функции.

Решение. Для размещения массива потребуется 100·200·6=120 000 байт. Используем динамический массив.

```
const m=100; n=200;
type row=array[1..n] of real;
    prow=^row;
    pmas=array[1..m] of prow;
var r:pmas; {r[i] – ссылка на i-ю строку. r[i]^ – содержимое i–й строки}
    i,j:integer; {индексы}

function getr(i,j:integer):real; {прочитать из массива элемент с индексами i, j}
        begin getr:=r[i]^[j]; end;

procedure putr(i,j:integer;x:real); {записать x в элемент массива i,j}
  begin r[i]^[j]:=x; end;

begin for i:=1 to m do getmem(r[i],sizeof(row));
            for i:=1 to m do
            begin writeln;
            for j:=1 to n do begin putr(i,j,random(10));
                                          write(getr(i,j),' ');
                                    end;
            end;
             for i:=1 to m do freemem(r[i],sizeof(row));
end.
```

Задача 4. *Списком* называется структура данных, каждый элемент которой связывается со следующим элементом с помощью указателя. Создать модуль для работы со списками.

Решение.

```
unit spisok;
interface
type datatype=real;
  lst_ptr-^lst_type;
  lst_type=record data:datatype;
                        next:lst_ptr;
          end;
procedure add_beg(var pp:lst_ptr;d:datatype);
procedure add_end(var pp:lst_ptr;d:datatype);
procedure pass(pp:lst_ptr);
```

```
procedure search(pp:lst_ptr;d:datatype;var p:lst_ptr);
procedure add_after(q:lst_ptr;d:datatype);
procedure del(var pp:lst_ptr;p:lst_ptr);
procedure searchb(pp:lst_ptr;d:datatype;var p:lst_ptr);
procedure add_pre(var t:lst_ptr;q:lst_ptr;d:datatype);
implementation
{добавление элемента в начало списка}
  procedure add_beg(var pp:lst_ptr;d:datatype);
var p:lst_ptr;
begin new(p);
    p^.data:=d;
    p^.next:=pp;
    pp:=p;
end;
{добавление элемента в конец списка}
procedure add_end(var pp:lst_ptr;d:datatype);
var p,q:lst_ptr;
begin p:=pp;
            if p=nil
            then begin new(p);
                              p^.data:=d;
                              p^.next:=nil;
                              pp:=p;
            end
             else begin while p^.next<>nil do p:=p^.next;
                               new(q);
                               q^.data:=d;
                               q^.next:=nil;
                               p^.next:=q;
                        end;
end;
{обход и печать списка}
procedure pass(pp:lst_ptr);
var p:lst_ptr;
begin p:=pp;
    while p<>nil do
    begin write(p^.data:1:0,' ');
                p:=p^.next;
    end;
    writeln;
end;
{поиск элемента, содержащего нужные данные,
pp – начало списка,
d – искомый элемент,
p – адрес нужного элемента}
procedure search(pp:lst_ptr;d:datatype;var p:lst_ptr);
begin p:=pp;
    while (p<>nil) and (p^.data<>d) do p:=p^.next;
end;
{добавление элемента после элемента с адресом q}
procedure add_after(q:lst_ptr;d:datatype);
var p:lst_ptr;
```

```
begin new(p);
            p^.next:=q^.next;
            p^.data:=d;
            q^.next:=p;
end;
{добавление элемента перед элементом с адресом q}
procedure add_pre(var t:lst_ptr;q:lst_ptr;d:datatype);
var p,s:lst_ptr;
begin p:=t;
  if p=nil then add_beg(t,d)
             else if q=nil then add_end(t,d)
                 else begin while p^.next<>q do p:=p^.next;
                                 new(s);
                                 s^.next:=q;
                                 s^.data:=d;
                                 p^.next:=s;
                      end;
end;
{удаление из списка элемента с адресом p}
procedure del(var pp:lst_ptr;p:lst_ptr);
var q:lst_ptr;
begin if p<>pp
            then begin q:=pp;
                            while q^.next<>p do q:=q^.next;
                            q^.next:=p^.next;
                      end
            else pp:=p^.next;
end;
{поиск элемента, большего заданного,
pp – начало списка,
d – искомый элемент,
p – адрес нужного элемента}
procedure searchb(pp:lst_ptr;d:datatype;var p:lst_ptr);
var f:boolean;
begin p:=pp;f:=false;
            while (p<>nil) and not f do
              if p^.data<=d
              then p:=p^.next
              else f:=true;
            if f then else p:=nil;
end;
end.
```

Задача 5. Используя модуль из задачи 16.2.4, создать и распечатать списки, вставляя элементы в начало и конец, затем создать упорядоченный список, если элементы для занесения в него генерируются случайным образом.

Решение.

```
uses spisok;
var j:integer; {элемент списка}
   i:integer; {индекс}
  t,              {указатель начала списка}
  s:lst_ptr;   {адрес нужного элемента}
begin t:=nil;
```

```
    for i:=1 to 9 do add_beg(t,i);
    pass(t);
    t:=nil;
    for i:=1 to 9 do add_end(t,i);
    pass(t);
    search(t,5,s);
    add_after(s,100);
    pass(t);
    t:=nill;
    for i:=1 to 10 do
    begin j:=random(10);
                searchb(t,j,s);
                if s=nil then add_end(t,j)
                else add_pre(t,s,j);
    end;
    pass(t);
end.
```

Задача 6. Одномерные массивы, двухмерные массивы, массивы больших размерностей, динамические массивы, записи, массивы записей, файлы – все они являются структурами данных. Далее рассматриваются и другие структуры данных. Для освоения методов работы с каждой конкретной структурой данных необходимо уметь решать перечисленные ниже классы задач.

К *первому* классу относятся задачи, в которых необходимо одинаково обработать каждый элемент структуры данных. Например:

1) ввести элементы структуры данных с клавиатуры или из файла;

2) вывести элементы структуры данных на экран или в файл;

3) найти сумму элементов структуры данных (каждый элемент добавляется к сумме);

4) найти количество элементов в структуре данных;

5) найти максимальный или минимальный элемент структуры данных (каждый элемент сравнивается с кандидатом на максимальное или минимальное значение);

6) найти количество (сумму, произведение и т. п.) элементов структуры, обладающих заданным свойством (четных, кратных *n*, больших среднего арифметического элементов структуры и т. п.);

7) просмотреть структуру данных парами (тройками, четверками и т. д.) и установить (подсчитать) свойства пары (оба элемента четные, первый элемент больше (меньше) второго, один положительный, другой отрицательный и т. п.);

8) решить перечисленные выше задачи для указанной части структуры данных (для заданного диапазона, для элементов с четным номером, для каждого большего заданного и т. п.).

Ко *второму* классу относятся задачи, в которых изменяется строение (порядок следования элементов) структуры данных. Например:

1) добавить новые элементы в начало, конец или в заданные места структуры данных (перед или после указанных элементов, так, чтобы не нарушилась упорядоченность структуры);

2) удалить указанные элементы из структуры данных (например, все четные);

3) изменить указанные элементы структуры данных;

4) изменить порядок следования элементов на обратный (перевернуть структуру);

5) сдвинуть элементы структуры указанным образом, в том числе циклически.

К *третьему* классу задач относятся задачи на совместную обработку нескольких однотипных структур данных, например двух одномерных массивов или одномерного массива и его части (подмассива). Например:

1) переписать структуру данных в другую (целиком или указанную часть (например, элементы с четными номерами), или элементы, обладающие заданным свойством, или в заданном порядке);

2) поменять местами части структуры, состоящие из нескольких элементов;

3) объединить несколько структур данных в одну, сохранив указанный порядок элементов или отобрав элементы, удовлетворяющие заданным свойствам;

4) переписать из одной структуры в другую те элементы, которые соответствуют заданным элементам третьей структуры. Например, из одного массива переписать в другой те элементы, для которых соответствующие элементы третьего массива равны единице (работа с маской).

К *четвертому* классу относятся задачи, в которых присутствует поиск. Например:

1) поиск элемента в неупорядоченной структуре (рассмотрите рекурсивный и нерекурсивный варианты);

2) поиск элемента в упорядоченной структуре;

3) проверить равенство структур (содержат одинаковые элементы в одном и том же порядке);

4) переписать из двух структур данных в третью те элементы, которые имеются хотя бы в одной структуре. Повторяющиеся элементы дважды не переписываются;

5) переписать из двух структур данных по одному разу в третью те элементы, которые присутствуют в обеих структурах;

6) переписать из двух структур данных в третью те элементы, которые присутствуют в первой, но отсутствуют во второй;

7) проверить, расположены ли два (n) заданных элемента рядом в структуре данных;

8) расположить элементы структуры в указанном порядке (отсортировать с использованием дополнительной структуры и без нее);

9) проверить, расположены ли элементы структуры в указанном порядке;

10) элементы структуры данных упорядочены и не повторяются. Добавить в нее новые элементы, не нарушая упорядоченности;

11) элементы структуры данных упорядочены и повторяются. Добавить в нее новые элементы, не нарушая упорядоченности.

К *пятому* классу относятся задачи, в которых используются несколько различных структур данных, причем для работы с каждой структурой данных приходится применять методы решения задач из предыдущих четырех классов. Например:

1) переписать элементы одной структуры в другую (все или специально указанные, обладающие заданным свойством);

2) сравнить содержимое заданных структур данных на равенство;

3) обработать некоторым образом каждый элемент первой структуры данных в соответствии с содержимым второй структуры;

4) переставить элементы первой структуры данных в соответствии с порядком элементов второй структуры данных;

5) соединить элементы двух различных структур данных в третью структуру либо друг за другом, либо в заданном порядке (по возрастанию, по убыванию, через один (два, три и т. д.) элемент), либо по заданному условию;

6) проверить все ли (заданные) элементы первой структуры встречаются во второй;

7) вывести элементы структур, встречающиеся хотя бы в одной структуре;

8) вывести элементы структур, встречающиеся одновременно в обеих структурах;

9) вывести элементы структур, встречающиеся во второй структуре, но отсутствующие в первой;

10) вывести элементы первой структуры, большие среднего арифметического второй структуры;

11) установить, что больше: максимальный элемент первой структуры или минимальный элемент второй структуры;

12) вывести элементы первой структуры, большие или равные заданному элементу второй структуры, если он там имеется;

13) проверить, имеется ли заданная часть первой структуры во второй;

14) проверить, можно ли получить порядок элементов, заданный первой структурой, переставив элементы второй структуры;

15) проверить, в какой структуре поиск заданного элемента осуществляется быстрее;

16) проверить, равны ли первый и последний элементы структур данных;

17) переставить местами элементы первой структуры так, чтобы они были больше соответствующих элементов второй структуры данных;

18) перечислить элементы, которыми отличаются заданные структуры данных;

19) перечислить элементы, по которым совпадают указанные структуры данных (без повторений, но с учетом порядка следования элементов);

20) вывести сначала по одному первому элементу структур, затем по два вторых, по два третьих и т. д. элементов структур данных;

21) вставить (удалить) в первую структуру все элементы, которые имеются во второй структуре.

К *шестому* классу относятся задачи, в которых одна структура данных моделируется с помощью другой. Например:

1) организовать обработку одномерного динамического массива как двухмерного, с количеством строк n и количеством столбцов m. Использовать для обращения к j-му элементу i-й строки формулу: $(i - 1)*m + j$;

2) с помощью одного одномерного массива смоделировать работу со списком;

3) с помощью двух одномерных массивов смоделировать работу со списком.

Задача 7. Нарисовать летящую птичку, используя технологию запоминания повторяющихся фрагментов изображения в динамической памяти.

Решение. Сначала нарисуем несколько (в приведенной программе – две) отличающихся друг от друга картинок с изображением птички. Каждую картинку сохраним в динамической памяти. Восстанавливая по очереди каждую картинку, задавая выдержку и стирая изображение, получим эффект полета птички на экране.

```
uses crt,graph;
var       mode:integer;
  driver:integer;
  err:integer;
  size:word;       {размер картинки}
  p,q:pointer;     {адреса в памяти, где хранятся образы картинок}
  x,y,     {координаты точки, где рисуется изображение}
  i,n,     {счетчик и количество рисуемых изображений}
  d:integer;       {время задержки}
begin
driver:=detect;
initgraph(driver,mode,'c:\tp7\bgi');
err:=graphresult;
if err<>0
then      begin writeln('Ошибка графики: ',grapherrormsg(err));
                  halt(1);
  end;
line(0,0,20,20);line(20,20,40,0);          {рисунок птички в первом положении}
```

```
size:=imagesize(0,0,40,20);     {определение размера рисунка}
getmem(p,size);          {получение места в куче для хранения рисунка}
getimage(0,0,40,20,p^);          {запоминание рисунка в отведенном месте}
  readln;
putimage(0,0,p^,xorput);         {стирание изображения первой птички}
line(0,0,30,20);line(30,20,60,0);        {рисунок птички во втором положении. Следующие операторы повторяются для второй птички}
size:=imagesize(0,0,60,20);
getmem(q,size);
getimage(0,0,60,20,q^);
  readln;
putimage(0,0,q^,xorput);
x:=10;y:=10;     {место восстановления изображения}
d:=1000;         {величина задержки}
n:=120;  {восстанавливаем n изображений}
for i:=1 to n do
begin if i mod 2=0       {если i четное, то восстанавливаем птичку в первом положении, иначе – во втором}
   then begin    putimage(x,y,p^,normalput);
{восстановили первый рисунок}
                 delay(d);
                 putimage(x,y,p^,xorput);
{стерли первый рисунок}
              end
   else begin putimage(x,y,q^,normalput);
{восстановили второй рисунок}
                 delay(d);
                 putimage(x,y,q^,xorput);
{стерли второй рисунок}
            end;
{изменяем координаты восстановления рисунков}
          x:=x+5;
          if x>getmaxx then x:=5;
          y:=y+5;
          if y>getmaxy then y:=5;
end;
closegraph;
end.
```

19. Отладка Паскаль-программ

Задача 1. Придумать тесты для отладки приведенной ниже программы для нахождения наибольшего общего делителя двух натуральных чисел.

```
{вычисление НОД двух натуральных чисел}
var      a,b:integer;     {исходные натуральные числа}
  x,y:integer;     {переменные, используемые для поиска НОД}
begin    write('Введите два натуральных числа ');
          readln(a,b);
          x:=a; y:=b;              { A }
          while x<>y do{ B }
                 if x>y            { C }
                 then x:=x-y       { D }
                 else y:=y-x;      { E }
          write(x);                { F }
```

end.

Решение. Для тестирования построим столько тестов, чтобы можно было пройти по каждой ветви программы хотя бы по разу. В ней имеется пять ветвей:

первая – A, B;
вторая – B, C;
третья – B, F;
четвертая – C, D, B;
пятая – C, E, B.

Путей здесь много. Примерами путей являются последовательности из следующих ветвей:

первый – 1, 3;
второй – 1, 2, 4, 3;
третий – 1, 2, 5, 3;
четвертый – 1, 2, 4, 2, 5, 3;
пятый – 1, 2, 4, 2, 4, 2, 4, 3;
шестой – 1, 2, 5, 2, 5, 2, 4, 3 и т. д.

Среди указанных путей есть такие, в которых используются не все ветви (это пути 1, 2, 3, 5), но есть и такие, в которых проходится каждая ветвь программы хотя бы по одному разу (это пути 4, 6).

Для построения теста выбранного пути построим сначала предикат пути, который будет принимать значение «истина», если в процессе исполнения программы реализуется данный путь, и «ложно» – при реализации других путей.

Для построения предиката пути проходим путь в обратном направлении и выписываем конъюнкцию всех условий, которые встречаются на этом пути. Если при обратном проходе пути осуществляется переход через оператор присваивания (переменная:=выражение), то в предикат пути вместо всех вхождений переменной из левой части оператора присваивания подставляется выражение из его правой части. Предикаты пути, построенные так, как описано выше, всегда выражаются в терминах констант и входных переменных. Чтобы пройти заданный путь во время исполнения программы, необходимо найти такие значения исходных данных, которые делают предикат пути истинным.

Рассмотрим процесс построения предиката для четвертого пути приведенной выше программы.

Вначале проходим ветвь 3 от F к B. При исполнении программы в блок F попадаем в случае ложности условия B, поэтому предикат этой ветви имеет вид P: X=Y.

Движемся дальше от B к C через E. При переходе оператора присваивания Y:=Y–X в предикате P все вхождения переменной Y заменим на выражение Y–X. Получаем P: X=Y–X или P: 2*X=Y. Поскольку E выполняется в случае ложности условия C, присоединяем его отрицание через конъюнкцию к P. P: (2*X=Y) and (X<=Y).

Движемся от C к B и присоединяем к P через конъюнкцию условие B (здесь оно должно быть истинным). P: (2*X=Y) and (X<=Y) and (X<>Y). В полученном предикате из первого члена конъюнкции 2*X=Y следует третий: X, не равный Y, поэтому третий член конъюнкции можно отбросить. Получим P: (2*X=Y) and (X<=Y).

Движемся от B к C через D. При переходе через оператор присваивания (X:=X–Y) все вхождения X заменяем выражением X–Y и присоединяем через конъюнкцию условие C (здесь оно истинно). P: (2*(X–Y)=Y) and (X–Y<=Y) или (2*X=3*Y) and (X<=2*Y).

Присоединяем условие C (здесь оно истинно). P: (2*X=3*y) and (x<=2*y) and (x<y). Анализируя полученный предикат, замечаем, что если истинен предпоследний член конъюнкции, то истинен и последний. Таким образом, последний член на истинность предиката не влияет, поэтому его можно отбросить. P: (2*X=3*Y) and (X<=2*Y).

Проходим ветвь 2 от C к B и к предикату через конъюнкцию присоединяем условие B (здесь оно истинно). Получаем P: (2*X=3*Y) and (X<=2*Y) and (X<>Y).

В полученном предикате вновь из первого члена конъюнкции следует третий, поэтому его можно отбросить. Получаем P: (2*X=3*Y) and (X<=2*Y).

Проходим ветвь 1 от B к A, при этом предикат пути не изменяется. Окончательно получаем P: (2*X=3*Y) and (X<=2*Y).

Для построения теста осталось подобрать значения X и Y так, чтобы предикат стал истинным. Это возможно, например,

при X=3, Y=2, получим ответ 1;
X=6, Y=4, получим ответ 2;
X=9, Y=6, получим ответ 3;
X=12, Y=8, получим ответ 4 и т. д.

Таким образом, представлены методики решения по двум из девятнадцати выделенных классов задач:

18. Указатели. Динамическая память. Структуры данных.
19. Отладка Паскаль-программ.

Этой статьей мы закончили знакомство с методикой быстрого обучения программированию на основе изучения классов задач.

Литература

1. *Аляев Ю. А., Козлов О. А.* Алгоритмизация и языки программирования Pascal, C++, Visual Basic. – М.: Финансы и статистика, 2002, 2004, 2007. 320 с.

2. *Аляев Ю. А., Гладков В. П., Козлов О. А.* Практикум по алгоритмизации и программированию на языке Паскаль. – М.: Финансы и статистика, 2004, 2007. 528 с.

3. *Аляев Ю. А.* Методика быстрого обучения программированию на основе изучения классов задач (1–3) // Образовательные ресурсы и технологии, 2015. № 1 (9). С. 3–14. http://www.muiv.ru/vestnik/pdf/pp/ot_2015_1_3-14.pdf.

4. *Аляев Ю. А.* Методика быстрого обучения программированию на основе изучения классов задач (4–5) // Образовательные ресурсы и технологии, 2015. № 2 (10). С. 3–16. http://www.muiv.ru/vestnik/pdf/pp/ot_2015_2_3-16.pdf.

5. *Аляев Ю. А.* Методика быстрого обучения программированию на основе изучения классов задач (6–7) // Образовательные ресурсы и технологии, 2015. № 3 (11). С. 3–20. http://www.muiv.ru/vestnik/pdf/pp/ot_2015_003_020.pdf.

6. *Аляев Ю. А.* Методика быстрого обучения программированию на основе изучения классов задач (8–10) // Образовательные ресурсы и технологии, 2015. № 4 (12). С. 26–43. http://www.muiv.ru/vestnik/pdf/pp/ot_2015_4_026-043.pdf.

7. *Аляев Ю. А.* Методика быстрого обучения программированию на основе изучения классов задач (11–15) // Образовательные ресурсы и технологии, 2016. № 1 (13). С. 8–20. http://www.muiv.ru/vestnik/pdf/pp/ot_2016_1_008-020.pdf.

8. *Аляев Ю. А.* Методика быстрого обучения программированию на основе изучения классов задач (16–17) // Образовательные ресурсы и технологии, 2016. № 3 (15). С. 3–14. http://www.muiv.ru/vestnik/pdf/pp/ot_3_2016_003-014.pdf.

Methods of the quick education to programming on base of the study of the classes of the problems (18–19)

***Yuri Alexandrovich Alyaev**, assistant professor, assistant professor of the pulpit of software of the computing machinery and automated systems, Perm military institute of internal troops of the MIA of Russia*

Is offered methods of the quick education to programming on base of the study of the classes of the problems, designed and using in practice in process of the education to programming student high school.

The Keywords: algorithm, program, programming language Pascal, array, pointers, dynamic memory, structures of data, debug.