19. ДЪРВОВИДНИ СТРУКТУРИ

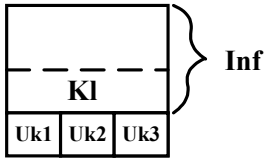
19.1. Общо описание. Основни понятия

Дървото е съвкупност от свързани компоненти, организирани в йерархична структура по своите ключови стойности.

Компоненти (възли) на дървото. Компонентите на дървото са динамични променливи от тип запис със следните полета (Фиг.19.1):

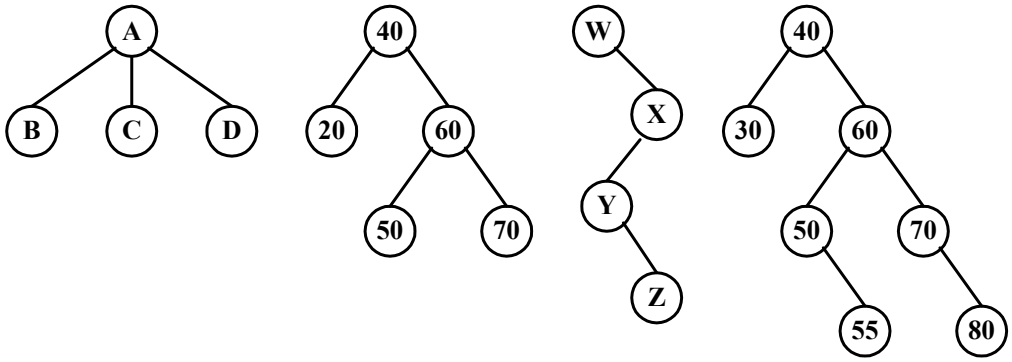
- *информационно поле* Inf - обикновено е от тип запис с полета, определени от конкретните цели, за които се създава дървото. Едно от полетата трябва да е уникално и то изпълнява ролята на ключ. Тук ще приемем, че информационното поле има само две полета: Ime и Kl, където Kl е ключ;

- *указателни полета* Uk1, Uk2,..., UkN, - указващи компонентите-наследници. Очевидно, за да е дърво динамичната структура, указателните полета трябва да са най-малко две.



Фиг.19.1 Компонента (възел) на дърво

На диаграмите възлите се означават с кръгчета. В кръгчето обикновено се дава само стойността на ключа на възела. На Фиг.19.2 са показани няколко дървовидни структури.

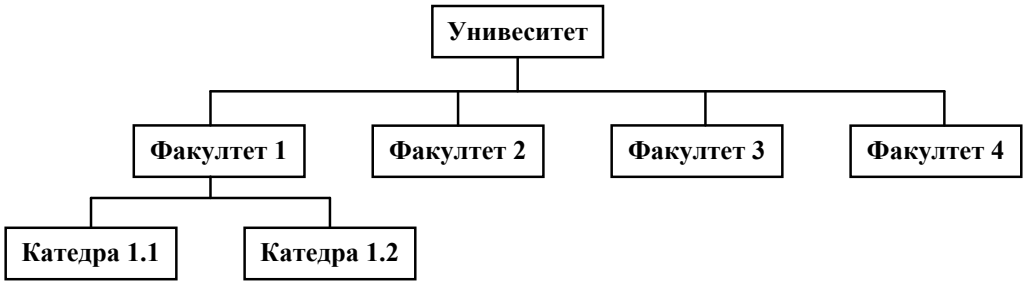


Фиг.19.2 Примерни дървовидни структури

Йерархичното структуриране е естествен, полезен и често срещан в практиката процес. Могат да се приведат много примери, потвърждаващи това. Различните правителствени, образователни, военни и стопански организации имат йерархична структура. На Фиг.19.3 е дадена примерна йерархична структура на някакъв университет.

Клон. Линиите, които свързват възлите се наричат ребра или клони. Те отразяват връзката между два възела, като между два възела може да има само една връзка. **Възли-родители, възли-наследници, възли-поколение**. Възелът A от Фиг.19. е родител на възлите B и C, ако те са указвани от указателни полета на

възела A. Възлите B и C са братя наследници на A. Всичките наследници на един възел образуват неговото поколение.



Фиг.19.3 Примерна йерархична университетска структура

Лист. Възел без наследници се нарича лист.

Празно дърво. Дърво, което не съдържа нито един възел, се нарича празно дърво.

Ориентирано дърво. Корен. Ориентираното дърво е празно дърво или дърво, запълнено по следния начин:

- компонентата, постъпила първа в дървото се приема за негов корен;
- останалите компоненти образуват **поддървета**, всяко от които е ориентирано дърво.

Обикновено под понятието *дърво* се разбира ориентирано дърво.

Път, насочен път, дължина на път. Маршрутът от един възел към друг се нарича път. В ориентираното дърво пътят е насочен път, т.е. движение е възможно само в посока от корена към листата. В този случай очевидно не съществува път между всяка двойка възли. Броят на възлите по пътя между два възела е дължина на пътя между тези възли.

Степен на възел. Тя се изразява с броя на наследниците на възела и е нула когато възелът е листо. Възел със степен по-голяма от нула е **вътрешен възел**.

Прародител и потмък. Всички възли след A, по пътя от възел A до възел P, от едно ориентирано дърво са потомци на възела A, а възелът A е техен прародител. Коренът е очевидно прародител на всички възли в дървото.

Поддърво. В ориентираното дърво всеки клон е поддърво, а всяко поддърво е ориентирано дърво.

Двоично дърво. В двоичното дърво всеки възел има не повече от два наследника.

Подредено двоично дърво. Подреденото двоично дърво е ориентирано дърво, при което всеки вмъкнат възел се подрежда йерархично според стойността на своя ключ (Фиг.19.5). То е познато и като **двоично дърво за търсене**, защото често се използва за реализация на бързо търсене. По-нататък в нашето изложение ще имаме пред вид двоичното дърво за търсене.

За да се организира двоично дърво е нужна една променлива от тип указател, която да указва корена (първата компонента) на дървото. С цел да се създадат известни удобства обикновено се ползва и една целочислена променлива за брояч на компонентите в дървото. Поради това, двоичното дърво ще присъства в примерите по-надолу като променлива от тип запис с две полета: указател към корена на дървото Uk и брояч на компонентите Br.

Тук ще разгледаме и основните операции с подредено дърво. Те са:

1. Инициализация на двоично дърво - привеждане на двоично дърво в състояние “празно двоично дърво”.
2. Добавене на компоненти.
3. Извличане на компоненти.
4. Изтриване на компоненти.
5. Обхождане на дърво.

19.2. Дефиниране на тип двоично дърво

```
Type
  TipKl=integer;      {Тип на ключа}
  TipInf=record       {Тип на информац. поле на комп. от дървото}
       Ime:string[25];
       Kl:TipKl
     end;
  TipUk=^TipEl;       {Тип на указател към комп. от дървото}
  TipEl=record        {Базов тип или тип на комп. от дървото}
       Inf:TipInf;    {Информац. поле на комп. от дървото}
       Lv,            {Указател към левия наследник}
       Ds:TipUk       {Указател към дясния наследник}
     end;
  TipStr=record       {Тип на структурата двоично дървото}
       Kor:TipUk;     {Указател към корена на дървото}
       Br:integer     {Брой на компонентите в дървото}
     end;
```

19.3. Дефиниране на двоични дървета (променливи от тип двоично дърво)

Двоичните дървета, нужни на програмата, се дефинират в раздела **Var**. Например с оператора:

```
Var
  DrvA, DrvB, DrvX : TipStr;
```

са дефинирани три дървета с имена съответно DrvA, DrvB и DrvX .

19.4. Инициализиране на двоично дърво

Под инициализиране на двоично дърво се разбира обявяване на двоичното дърво за празно, а това се постига като се присвои нулева стойност на брояча на компонентите и стойност **nil** на указателя към корена на дървото. На Фиг.19.4 са показани две инициализирани дървета.

Фиг.19.4 Инициализирани дървета

Дървета могат да се инициализират чрез следната процедура, където Str е фиктивен параметър, който се замества с името на инициализираното двоично дърво:

```
Procedure Init(Var Str:TipStr);
  Begin
    Str.Kor:=Nil; Str.Br:=0;
  End;
```

19.5. Добавяне на компонента към подредено двоично дърво

Компонентата, добавяна към подредено двоично дърво, трябва да попадне на място, отредено от самата йерархия на компонентите. Това място се търси както следва:

1. Ако ключовата стойност на новата компонента е по-голяма от ключовата стойност на корена, мястото й е в дясното поддърво, иначе мястото й е в лявото поддърво.

2. Ако ключовата стойност на новата компонента е по-голяма от ключовата стойност на корена на поддървото, мястото му е в неговото дясно поддърво, иначе мястото й е в неговото ляво поддърво. Тази операция се повтаря докато се окаже, че мястото на компонентата е в празно поддърво. Тогава компонентата се добавя като първа компонента (корен) в това празно поддърво.

Към двоично дърво може да се добави нова компонента например чрез функцията DobPoKl, дадена по-долу, където Str и X са фиктивни параметри, които се заместват съответно с името на двоично дърво и променливата, чиято стойност става стойност на информационното поле на новата компонента.

Във функцията DobPoKl е вложена рекурсивната функция Dobavi с два фиктивни параметъра: Kor - указател към корена на дървото и X - информационно поле на новата компонента.

```
Function DobPoKl(Var Str:TipStr;X:TipInf):boolean;
  Procedure Dobavi(Var Kor:TipUk;X:TipInf);
    Begin
      If Kor=Nil
      then begin     {Добавяне в празно дърво/поддърво}
        New(Kor); Kor^.Inf:=X; Kor^.Lv:=Nil; Kor^.Ds:=Nil
      end
      else  if X.Kl <= Kor^.Inf.Kl     {Търсене на празно поддърво}
            then Dobavi(Kor^.Lv,X)
            else Dobavi(Kor^.Ds,X)
    End;
  Begin
    If MaxAvail>=SizeOf(TipEl)
    then begin
       Dobavi(Str.Kor,X);
       Str.Br:=Str.Br+1;
       DobPoKl:=true
    end
    else DobPoKl:=false
  End;
```

19.6. Търсене на указател към компонента, съдържаща зададен ключ

Търсене на указател към компонента от двоично дърво, съдържаща зададена ключова стойност се налага, когато трябва да извлечем или изтрием компонента. Това търсене е задача на следващата процедура. Тя връща намерения указател или **nil**, ако в дървото няма компонента с посочената ключова стойност. В процедурата Kor е указател към корена на дървото, Trs - указател към търсената компонента, Rod - указател към родителя на търсената компонента.

```
Procedure TrsEl( Kor:TipUk;Kl:TipKl;Var Trs,Rod:TipUk);
  Begin
    If Kor=nil
    then Trs:=nil                    {Дървото/поддървото е празно}
    else begin
      If Kl<Kor^.Inf.Kl
      then begin                     {Търсене в лявото поддърво}
        Rod:=Kor;
        TrsEl(Kor^.Lv,Kl,Trs,Rod)    {Рекурсия}
      end
      else  if Kl>Kor^.Inf.Kl
          then begin                 {Търсене в дясното поддърво}
             Rod:=Kor;
             TrsEl(Kor^.Ds,Kl,Trs,Rod)     {Рекурсия}
          end
          else Trs:=Kor;             {Този е търсеният указател}
    end
  End;
```

19.7. Извличане на компонента по зададена стойност на ключовото поле

От двоично дърво може да се извлече компонента например чрез функцията IzvlPoKl, дадена по-долу. Фиктивните параметри се заместват както следва: Str - с името на двоичното дърво, Kl - със стойността на ключовото поле на компонентата, чието информационно поле трябва да се извлече и X - с променливата, която извлича информационното поле. Във функцията са дефиирани указателите: Izvl - указател към извлечената компонента и RodIzvl -указател към родителя на извлечената компонента.

Функцията IzvlPoKl използва функцията TrsUk, дадена по-горе, за да намери адреса на компонентата, подлежаща на извличане.

```
Function IzvlPoKl(Str:TipStr;Kl:TipKl;Var X:TipInf):boolean;
  Var
    Izvl,RodIzvl:TipUk;
  Begin
    TrsEl(Str.Kor,Kl,Izvl,RodIzvl);{Намиране указателя към търсената компонента}
    If Izvl=nil
    then IzvlPoKl:=false     {Търсената компонента не е намерена }
    else begin               {Търсентата компонента е намерена}
      X:=Izvl^.Inf;          {Извличане}
```

```
      IzvlPoKl:=true
    end
  End;
```

Функцията IzvlPoKl връща стойност true, когато е извлечено информационното поле на компонентата с посочен ключ, и стойност false, когато в дървото не е намерена компонента с посочената ключова стойност.

19.8. Изтриване на компонента със зададена стойност на ключовото поле

От двоично дърво от декларирания тип може да се изтрие компонента например чрез функцията IztrPoKl, дадена по-долу, където Str и Kl са фиктивни параметри. При обръщение към функцията те се заместват съответно с името на двоичното дърво и стойността на ключовото поле на компонентата, която трябва да се изтрие.

Изтриването на компоненти от двоично дърво става на две стъпки:

1. Намира се адресът на компонентата, която трябва да се изтрие. За тази цел се използва функцията TrsEl, която разгледахме по-горе.

2. Изтрива се компонентата с намерения адрес. Самото изтриване се предхожда от анализ относно вида на компонентата (мястото на компонентата в йерархията), защото изтриващите операции зависят от мястото на компонентата в йерархията.

Според мястото на компонентата в йерархията тя може да бъде:

а) Лист, който може да е:

- лист-корен;
- ляв лист на вътрешна компонента. За дървото от Фиг.19.5 такива листа са: 28, 50 и 70;
- десен лист на вътрешна компонента. За дървото от Фиг.19.5 такива листа са: 15, 39 и 80;

б) Корен или вътрешна компонента само с ляв наследник. За дървото от Фиг.19.5 такава компонента е 60;

в) Корен или вътрешна компонента само с десен наследник. За дървото от Фиг.19.5 такава компонента е 11;

г) Корен или вътрешната компонента с два наследника. За дървото от Фиг.19.5 такива компоненти са: 46, 21, 33, 62, 75. В този случай изтриваната компонента се замества с най-дясната в нейното ляво поддърво или с най-лявата в нейното дясното поддърво. В подпрограмата е прието заместваща да е най-дясната в лявото поддърво. След като се замести изтриваната, изтрива се заместващата - тя може да бъде листо или вътрешна, но само с ляв наследник.

Функцията IztrPoKl връща стойност true, когато е изтрила компонентата с посочената ключова стойност, и стойност false, когато не е намерила компонентата с посочената ключова стойност.

```
Function IztrPoKl(Var Str:TipStr;Kl:TipKl):boolean;
  Var
    Zam,     {Указател към заменящата компонента}
    RodZam,          {Указател към родителя на заменящата компонента}
```

```
    Pom:TipUk;      {Помощен указател}
    BrNasl:byte;    {Брой на наследниците на изтриваната компонента}
Begin
  TrsEl(Str.Kor,Kl); {Търсене на компонента за изтриване}
  If Trs=nil
    then begin
      IztrPoKl:=false;
      Exit
    end;{Не е намерена}
{Преброяване наследниците на изтрив. компонента}
  BrNasl:=0;
  If (Trs^.Lv<>nil) then BrNasl:=BrNasl+1;
  If (Trs^.Ds<>nil) then BrNasl:=BrNasl+1;
  Case BrNasl of
    0: begin          {Изтриване на листо}
       If Trs=Str.Kor          {Корен ли е изтриваното листо?}
         then Str.Kor:=nil   {Да. Изтриване на листо-корен}
         else if Trs=RodTrs^.Lv     {Не. Изтриване на листо-некорен}
              then RodTrs^.Lv:=nil {Листото е ляв наследник}
              else RodTrs^.Ds:=nil;{Листото е дес. наследник}
       Dispose(Trs)
       end;
    1:begin           {Изтриване на компонента с един наследник}
       If Trs^.Lv=nil
         then begin {Наследникът е десен}
              Pom:= Trs^.Ds;
              Trs^:= Trs^.Ds^;{Наследникът замества изтривания}
            end
         else begin {Наследникът е ляв}
              Pom:= Trs^.Lv;
              Trs^:= Trs^.Lv^;{Наследникът замества изтривания} end;
              Dispose(Pom)
            end;
    2:                        {Изтриване на комп. с два наследника}
       If Trs^.Lv^.Ds=nil    {Има ли левият наследник десен клон?}
         then begin{Няма десен клон. Заместващ е левият наследник}
              Pom:= Trs^.Lv;
              Trs^.Inf:= Trs^.Lv^.Inf;
              Trs^.Lv:= Trs^.Lv^.Lv;
              Dispose(Pom)
            end
         else begin {Левият наследник има десен клон.
       Заместващ е най-десният в този клон}
              RodZam:= Trs^.Lv;
              Zam:= RodZam^.Ds;
              While Zam^.Ds<>nil do
                begin
                  RodZam:= Zam;
                  Zam:= Zam^.Ds
                end;
              Trs^.Inf:= Zam^.Inf;
```

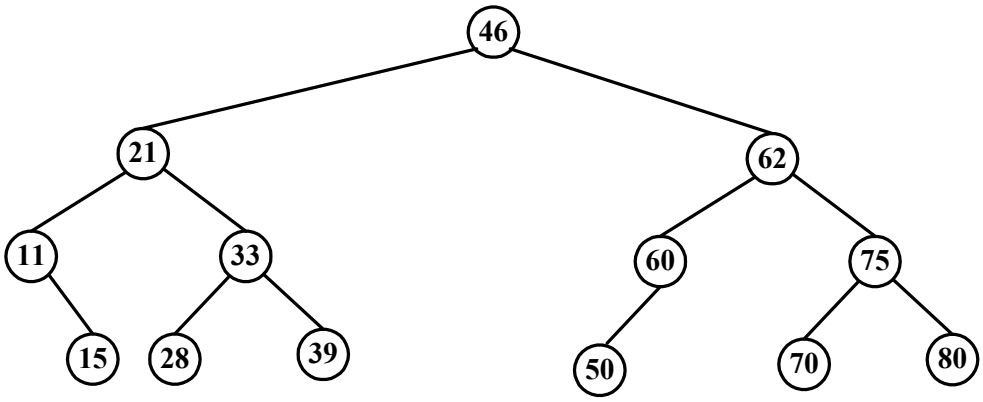
```
                RodZam^.Ds:= Zam^.Iv;
                Dispose( Zam)
              end
  end;{Край на оператора Case}
  IztrPoKl:=true;
  Str.Br:=Str.Br-1
End;
```

19.9. Обхождане на двоично дърво

Операцията обхождане на двоичното дърво представлява последователно посещение на всички възли с цел осъществяване на определена обработка на данните във всеки възел.

Обработката може да представлява извеждане на екрана на стойността на ключа, извеждане на екрана на цялото информационно поле на възела, прехвърляне на информацията от дървото в масив, в линеен списък или във файл и др.

Всяко обхождане на дървото започва от корена му. Според реда на посещение на възлите, обхождането може да бъде *обхождане в дълбочина* или *обхождане в широчина*. При обхождане в дълбочина се напредва до възможен край на дървото (т.е. към листата) и след това следва връщане обратно към предишния посетен възел. Следва ново напредване, ново връщане и т.н. При обхождане по широчина след корена се посещават от ляво на дясно всички възли от първо ниво, след това пак от ляво на дясно се посещават всички възли от второто ниво и т.н., т.е. обхождането по широчина всъщност е обхождане по нива.



Фиг.19.5 Подредено двоично дърво.

Съществуват шест начина за обхождане в дълбочина.

1. Обхождане в низходящ ред. То може да се осъществи по два начина:

1.1. Обхождане в реда КЛД;
- обработка на корена (К);
- обхождане на лявото поддърво(Л);
- обхождане на дясното поддърво (Д)

1.2. Обхождане в реда КДЛ;
- обработка на корена;
- обхождане на дясното поддърво;
- обхождане на лявото поддърво

При обхождане на дървото, показано на Фиг.19.5, по начина 1.1 и 1.2, съпроводено с извеждане на ключовата стойност във всеки посетен възел, ще получим съответно:

КЛД: 46, 21, 11, 15, 33, 28, 39, 62, 60, 50, 75, 70, 80
КДЛ: 46, 62, 75, 80, 70, 60, 50, 21, 33, 39, 28, 11, 15

2. Обхождане в смесен ред. То може да се осъществи по два начина:

2.1. Обхождане в реда ЛКД;
- обхождане на лявото поддърво;
- обработка на корена;
- обхождане на дясното поддърво

2.2. Обхождане в реда ДКЛ;
- обхождане на дясното поддърво;
- обработка на корена;
- обхождане на лявото поддърво

При обхождане на дървото, показано на Фиг.19.5, по начина 2.1 и 2.2, съпроводено с извеждане ключовата стойност във всеки посетен възел, ще получим съответно:

ЛКД: 11, 15, 21, 28, 33, 39, 46, 50, 60, 62, 70, 75, 80
ДКЛ: 80, 75, 70, 62, 60, 50, 46, 39, 33, 28, 21, 15, 11

3. Обхождане във възходящ ред. То може да се осъществи по два начина:

3.1. Обхождане в реда ЛДК;
- обхождане на лявото поддърво;
- обхождане на дясното поддърво;
- обработка на корена.

3.2. Обхождане в реда ДЛК;
- обхождане на дясното поддърво;
- обхождане на лявото поддърво;
- обработка на корена.

При обхождане на дървото, показано на Фиг.19.5, по начина 3.1 и 3.2, съпроводено с извеждане ключовата стойност във всеки посетен възел, ще получим съответно:

ЛДК: 15, 11, 28, 39, 33, 21, 50, 60, 70, 80, 75, 62, 46
ДЛК: 80, 70, 75, 50, 60, 62, 39, 28, 33, 15, 11, 21, 46

Както се вижда от примерите двата начина на низходящо обхождане (ЛКД и ДКЛ) водят до подреждане съответно в нарастващ и намаляващ ред.

По-долу следват процедури за обхождане на двоично дърво с цел извеждане съдържанието му на екрана.

```
Procedure Print(Str:TipStr);
  Var KodPr:char; Kl:TipKl;
{Обхождане в ред "корен, ляво поддърво, дясно поддърво" }
  Procedure PrKLD(Kor:TipUk);
    Begin
      If Kor<>Nil
      then begin
        WriteInf(Kor^.Inf);
        PrKLD(Kor^.Lv);
        PrKLD(Kor^.Ds)
      end;
    End;
{Обхождане в ред "корен, дясно поддърво, ляво поддърво" }
  Procedure PrKDL(Kor:TipUk);
    Begin
    If Kor<>Nil
      then begin
           WriteInf(Kor^.Inf);
           PrKDL(Kor^.Ds);
```

```
            PrKDL(Kor^.Lv)
            end;
    End;
{Обхождане в ред "дясно поддърво, корен, ляво поддърво"}
Procedure PrDKL(Kor:TipUk);
    Begin
    If Kor<>nil
      then begin
          PrDKL(Kor^.Ds);
          WriteInf(Kor^.Inf);
          PrDKL(Kor^.Lv);
          end
    End;
{Обхождане в ред "дясно поддърво, ляво поддърво, корен"}
Procedure PrDLK(Kor:TipUk);
    Begin
      If Kor<>nil
        then begin
              PrDLK(Kor^.Ds);
              PrDLK(Kor^.Lv);
              WriteInf(Kor^.Inf)
            end
    End;
{Обхождане в ред "ляво поддърво, корен, дясно поддърво"}
  Procedure PrLKD(Kor:TipUk);
    Begin
      If Kor<>nil
      then begin
        PrLKD(Kor^.Lv);
        WriteInf(Kor^.Inf);
        PrLKD(Kor^.Ds);
      end
    End;
{Обхождане в ред "ляво поддърво, дясно поддърво, корен"}
  Procedure PrLDK(Kor:TipUk);
    Begin
      If Kor<>nil
      then begin
        PrLDK(Kor^.Lv);
        PrLDK(Kor^.Ds);
        WriteInf(Kor^.Inf)
      end
    End;
  Begin
    If Str.Kor=nil
    then begin
      Writeln('Дървото е празно');
      Exit
    end;
```

```
  Repeat
    ClrScr;
    GotoXY(25, 9);Write('РЕД НА ОБХОЖДАНЕ НА ДЪРВОТО:');
    GotoXY(30,10);Write('1 - низходящ - КЛД');
    GotoXY(30,11);Write('2 - смесен     - ЛКД');
    GotoXY(30,12);Write('3 - възходящ - ЛДК');
    GotoXY(30,13);Write('4 - смесен      - ДКЛ');
    GotoXY(30,14);Write('5 - възходящ - ЛДК');
    GotoXY(30,15);Write('6 - възходящ - ДЛК');
    GotoXY(25,16);Write('Изберете ред на обхождане или 0: ');
    Repeat
      KodPr:=ReadKey
    Until KodPr in ['0'..'3'];
    Writeln;
    Case KodPr of
      '1':begin PrKLD(Str.Kor);Readln end;
      '2':begin PrLKD(Str.Kor);Readln end;
      '3':begin PrLDK(Str.Kor);Readln end;
      '4':begin PrDKL(Str.Kor);Readln end;
      '5':begin PrLDK(Str.Kor);Readln end;
      '6':begin PrDLK(Str.Kor);Readln end;
    end
  until KodPr='0'
End;
```