|  |  |
| --- | --- |
| program ob1;{самая пpостая без наследования}  uses max\_crt;  type  t11=object  a1:integer;{поля данных должны объявляться перед методами }  a2:char;  procedure Init(pa1:integer;pa2:char);{метод для инициализации полей данных объекта}  procedure pr;{метод для вывода полей данных}  end;  procedure t11.Init(pa1:integer;pa2:char);  Begin  a1:=pa1;a2:=pa2;  writeln(’\*\*\*\*\*’);  writeln('a1=',a1,’ a2=’,a2);  end;  procedure t11.pr;  begin  writeln('from pr’,’a1=’,a1,’ a2=’,a2);  end; | var  o1,o2:t11;  Begin  clrscr;  o1.init(10,’A’);{инициализация полей данных объекта о1}  o2.init(5,’C’);{ инициализация полей данныхобъекта о2}  o2.pr;{вывод полей данных объекта о2}  writeln(’o1.a1=’,o1.a1,' o2.a1=',o2.a1);{ вывод полей данных объектов o1 и о2}  end.  В результате выполнения этой программы на экран будут выведены строки:  \*\*\*\*\*  a1=10  a2=A \*\*\*\*\*  a1=5 a2=C  from pr a1=5 a2= C  o1.a1=10 02.a2=5  **имена полей данных объекта не должны совпадать с именами формальных параметров** у методов объекта. |

|  |  |
| --- | --- |
| program ob11;{простое наследование метода p1}  uses max\_crt;  type  t1=object  a:integer;  procedure p1(a1:integer);  end;  t2=object(t1)  b:string;  procedure p(b1:string);  end;  procedure t1.p1(a1:integer);  Begin  a:=a1 ;  writeln('a=',a);  end;  Procedure t2.p(b1:string);  Begin  b:=b1;  a:=50;  writeln('o2.a=',a);  writeln('o2.b=',b);  end; | var  o1:t1;o2:t2;  Begin  clrscr;  o2.p1(70);{в поле o2.a записали число 70}  o1.p1(100);{в поле o1.a записали число 100}  o2.p('\*\*\*');{в поле o2.a записали 50, а в поле o2.b –’\*\*\*’}  end.  В этом примере потомок типа t2 наследует от предка типа t1 поле данного **a** , а также процедуру p1, которая работает с полем o1.a или o2.a в зависимости от того, из какого объекта эта процедура запускается.  В результате работы этой программы на экран выводятся строки:  a=70  a=100  O2.a=50  O2.b=\*\*\* |

|  |  |
| --- | --- |
| Program ob12{переопределение методов}  uses max\_crt;  type  t1=object  a:integer;  procedure p1;  function f1(a1:string):string;  end;  t2=object(t1)  function f1:string;  end;  Function t1.f1;  Begin  f1:=a1 ;  end;  Procedure t1.p1;  Begin  writeln(f1('==='))  end;  Function t2.f1:string;{эту функцию предок переопределяет }  Begin  f1:='function t2.f1'  end;  var o1:t1;o2:t2;s:string;  Begin clrscr;  o1.p1; | o2.p1;  s:=o1.f1('ddd');{используется функция предка}  writeln('s=',s);  s:=o2.f1;{используется функция f1 потомка}  writeln('s=',s);  end.  1. При выполнении оператора o1.p1 осуществляется поиск метода p1 в объекте o1 ,он там обнаруживается и запускается, и запускается именно его функция f1, которая принимает значение ‘= = =’.  2.При вызове метода o2.p1 вначале этот метод ищется в объекте o2, его там нет. Тогда начинается поиск этого метода у предка , и он там обнаруживается и запускается. Но мы уже находимся в теле предка , поэтому запускается снова не метод потомка , а метод предка f1. Поэтому снова будет выведена строка на экран ‘= = = ‘ , а не ‘function t2.f1’, как бы должно было выводиться , поскольку метод p1 запускался из потомка o2.  будут выведены строки:  = = =  = = =  s=ddd  s=function t2.f1 |

|  |  |  |
| --- | --- | --- |
| Методы описываются как **виртуальные , если** **у предка и потомка имена методов и списки параметров совпадают, а тела методов различны**  **program ob2;{с виртуальными методами без параметров}**  **uses max\_crt;**  **{сделай попытку без virtual и сpавни эти ваpианты}**  **type**  **t1=object**  **a1:string;**  **procedure p1;**  **function f1:string;virtual;**  **constructor Init;**  **end;**  **t2=object(t1)**  **a2:string;**  **function f1:string;virtual;**  **constructor Init;**  **end;**  **Procedure t1.p1;**  **Begin**  **writeln(f1);**  **end;**  Function t1.f1:string;  Begin  f1:='+++';  a1:='tt';  writeln('a1=',a1);  end;  Constructor t1.Init;  Begin {var is absent}  end;  Function t2.f1:string;  Begin  f1:='----';  a1:='\*\*\*';a2:='!!!';  writeln('a1=',a1,' a2=',a2);  end;  Constructor t2.Init;  Begin  end;  var  o1:t1;o2:t2; {o1-предок, o2-потомок}  Begin  clrscr; | o1.init; {запуск конструктора предка}  o2.init; {запуск конструктора потомка}  o1.p1;{здесь будет использоваться метод f1 предка}  o2.p1;{здесь будет использоваться метод f1 потомка}  writeln(o2.f1,' ',o1.f1);  end.  Основные особенности использования виртуальных методов состоят в следующем:  1.Если в описании типа объекта есть виртуальные методы, то **обязательно должна быть специальная процедура, называемая Constructor.** Эта процедура позволит в дальнейшем правильно использовать виртуальные методы. Она может содержать любой список параметров, может быть без параметров , а может быть с пустым телом ,т.е. не содержать никаких операторов внутри , как в нашем примере. Но присутствовать она должна обязательно.  2.**Перед запуском виртуального метода объекта должен быть запущен Constructor**,поскольку без этого действия не определен путь к виртуальному методу объекта.  3.**Если метод не является виртуальным** (является статическим), связь его с объектом осуществляется компилятором на основании описания объекта. При этом говорят , что **происходит “раннее связывание” метода с объектом**. | **В случае наличия виртуальных методов** в описании объекта их связь с объектом осуществляется на этапе исполнения программы при запуске программы Constructor, после чего появляется возможность использовать этотвиртуальный метод. При этом **используют термин “позднее связывание (метода с объектом)”.**  **Если в тексте всюду оставить слова virtual, то получим результат:**  **а1=tt**  **+++**  **a1=\*\*\*a2=!!!**  **----**  **a1=\*\*\*a2=!!!**  **----a1=tt**  **+++**  **Если закомментировать слова Virtual в тексте этой программы , то получим:**  **a1=tt {при обращении к o1.p1}**  **+++**  **a1=tt {при обращении к o2.p1}**  **+++**  **a1=\*\*\* a2=!!! {при выполнении оператора wrileln в головной программе}**  **---- a1=tt**  **+++** |

|  |  |
| --- | --- |
| program ob13;{простое переопределение метода P1}  uses max\_crt;  type  t1=object  a:integer;  procedure p1(a1:integer);{для инициализации поля a}  function f:integer;  end;  t2=object(t1)  b:string;  procedure p1(b1:string);{для инициализации полей a и b}  function f1:string;  end;  procedure t1.p1(a1:integer);  Begin  a:=a1 ;  end;  Function t1.f:integer;  begin  f:=a  end;    Function t2.f1;  begin  f1:=b  end;  Procedure t2.p1(b1:string);{список параметров отличается от списка в методе t1.p1}  Begin {тело отличается от тела t1.p1}  b:=b1;  a:=50;  end; | var  o1:t1;o2:t2;  Begin  clrscr;  o2.p1('+++');  o1.p1(100);  writeln('o1.f=',o1.f);  writeln('o2.f=',o2.f);  writeln('o2.f1=',o2.f1);  end.  Из описания типов и переменных следует, что у объекта o1 есть одно поле данного **a** и методы **P1** и **f,** а у объекта o2 есть поля данных **a** и **b** и методы **p1**(переопределяемый по отношению к p1 предка),**f** (унаследованный от предка) и свой метод **f1**.  При вызове метода o2.p1 переменные приобретают значения: a=50 , b=’+++’ .  При вызове метода o1.p1 поле a объекта o1 приобретает значение 100.  При вызове o1.f в операторе writeln на место имени f записывается значение o1.a =100.  При вызове метода o2.f в операторе writeln на место имени f записывается значение o2.a=50.  При вызове метода o2.f1 на место имени f1 записывается значение o2.b=’+++’.  Поэтому в результате исполнения этой программы на экран будут выведены строки:  O1.f=100  O2.f=50  O2.f1=+++ |

|  |  |
| --- | --- |
| program ob23; {Вызов методов предка из потомка}  uses max\_crt;  type  t1=object  function f1:string;  end;  t2=object(t1)  function f1:string;{метод переопределяет метод предка}  procedure p2(a1:string);  end;  Function t1.f1:string;  Begin  f1:='---'  end;  Function t2.f1:string;  Begin  f1:='+++++'  end;  procedure t2.p2(a1:string);  begin  a1:=inherited f1;{вызывается не свой метод, а метод ближайшего предка}  writeln('a1=',a1) ;{вывод значения string , сформированного в теле метода предка}  writeln('a1= ', f1);(вызывается свой метод)  end; | var  o1:t1;o2:t2;s:string;{используются статические объекты}  Begin  clrscr;  o2.f1;{запуск метода потомка}  o2.p2(s) ;{Внутри p2 вначале вызывается метод предка f1, а затем свой метод f1}  writeln('o1.f1=',o1.f1 ) {запуск метода предка}  end.  Слово **inherited** используется для указания на то , что надо обратиться к методу ближайшего предка , т.е. к методу t1.f1. Поэтому внутри метода t2.p2 предусмотрен вызов метода предка t1.f1 , а затем своего метода t2.f1. В результате работы этой программы на экран выводятся строки:  a1=---  a1=+++++  o1.a1=--- |

|  |  |
| --- | --- |
| program ob32;{виртуальные методы с параметрами}  uses max\_crt;  type  t1=object  procedure p1;  function f1(a1:string):string;virtual;  constructor Init;  end;  t2=object(t1)  b:byte;  function f1(a1:string):string;virtual;  constructor Init;  end;  Procedure t1.p1;  Begin  writeln(f1('= = ='))  end;  Function t1.f1(a1:string):string;  Begin  f1:=a1  end;  Constructor t1.Init;  Begin {var is absent}  end;  Function t2.f1(a1:string):string;  Begin  f1:=a1;  b:=ord(a1[0]);  writeln('b=',b)  end;  Constructor t2.Init;  Begin  end;  var  o1:t1;o2:t2;s:string; | Begin  clrscr;  o1.init;{запуск конструкторов}  o2.init;  o1.p1;{будет вызван метод f1 предка}  o2.p1;{ будет вызван метод f1 потомка }  s:=o1.f1('uuu');{вызов метода предка}  writeln('s=',s);  s:=o2.f1('ddd');{вызов метода потомка}  writeln('s=',s);  writeln('= = =’,o2.f1('\*\*\*'));{вызов метода потомка}  writeln('----',o1.f1('^^^'));{вызов метода предка}  end.  В этой программе при исполнении оператора writeln головной программы , в котором вызывается метод t2.f1, вначале исполняется оператор writeln, имеющийся внутри тела t2.f1 , а затем оператор writeln головной программы.В результате исполнения этой программы на экран будут выведены строки:  = = =  b=3  = = =  s=uuu  b=3  s=ddd  = = = = b=3  \*\*\*  ----^^^  Таким образом, полиморфизм - это не только совпадение имен методов в иерархии объектов , но и возможность по-разному их исполнять на разных уровнях иерархии. |

|  |  |
| --- | --- |
| Program ob35;{инициализация полей данных}  uses max\_crt;  type  t1=object  a1:integer;  a2:char;  procedure p1(pa1:integer;pa2:char);  Constructor Init(pa1:integer;pa2:char);  procedure pout; virtual;  end;  t2=object(t1)  c: string;  procedure p3(pa3:string);  procedure pout; virtual;  end;  procedure t1.p1(pa1:integer;pa2:char);  begin  a1:=pa1;a2:=pa2;  end;  Constructor t1.Init(pa1:integer;pa2:char);  Begin  a1:=pa1;a2:=pa2;  end; procedure t1.pout;  Begin  Writeln ('t1.pout',' a1=',a1,' a2=',a2);  end;  procedure t2.p3(pa3:string);  begin  c:=pa3  end;  procedure t2.pout;  Begin  writeln('t2.pout',' a1=',a1,' a2=',a2,' c=',c);  end;  var  o1:t1;o2:t2;  begin  clrscr;  o1.Init(5,'f');{инициализация с помощью конструктора}  o1.pout; {вывод своей виртуальной процедурой}  o2.Init(6,'g');{запуск конструктора, наследуемого от объекта o1} | o2.p1(7,'h');{повторная инициализация полей o2.a1 и o2.a2}  o2.p3('\*\*\*'); {инициализация поля o2.c}  o2.pout; {вывод полей объекта o2}  o1.a1:=100;  writeln('o1.a1=',o1.a1)  end.  Отметим важные особенности этого текста.  1. У потомка отсутствует свой конструктор , так как он наследует его от предка .  2. Инициализацию данного можно проводить как с помощью метода t1.p1, так и с помощью конструктора , который можно наследовать и обращаться к нему из потомка.  3. Без запуска конструкторов обоих объектов нельзя обращаться к их виртуальным методам pout.  4. Инициализация полей o2.a1 и o2.a2 проводится дважды (первый раз с помощью конструктора предка, а второй раз с помощью своего метода p1), поэтому на экране не увидим значений 6 и ‘g’.  5. В последних двух операторах программы поле o1.a1 инициализируется непосредственно присвоением, и при выводе происходит непосредственное обращение к этому полю , как в случае **record.**  Однако **идеология объектно-ориентированного программирования предполагает все действия по вводу и выводу значений полей данных осуществлять с помощью собственных методов**. В результате исполнения этой программы на экран будут выведены строки:  t1.pout a1=5 a2=f  t2.pout a1=7 a2=h c=\*\*\*  O1.a1=100 |

|  |  |
| --- | --- |
| Program ob36;{инициализация полей данных внешней процедурой}  uses max\_crt;  type  t1=object  a1:integer;  a2:char;  Constructor Init;  procedure pout;virtual;{для вывода своих полей данных}  end;  t2=object(t1)  c:string;  {процедура p3 обеспечивает инициализацию данных этого типа объектов}  procedure p3(pa1:integer;pa2:char;pa3:string);  procedure pout;virtual;{для вывода своих полей данных}  end;  Constructor t1.Init;  Begin  end;    procedure t1.pout;  Begin  writeln('t1.pout',' a1=',a1,' a2=',a2);  end;  procedure t2.p3(pa1:integer;pa2:char;pa3:string);  begina1:=pa1;a2:=pa2; c:=pa3  end;  procedure t2.pout;  Begin  writeln('t2.pout',' a1=',a1,' a2=',a2,' c=',c);  end;  {начало внешней процедуры инициализации полей объектов типа t1}  procedure p1(var ob:t1; pa1:integer; pa2:char);  begin  with ob do  begin  a1:=pa1;a2:=pa2;  end;  end; {конец внешней процедуры инициализации полей объектов типа t1} | var  o:array[1..3] of t1;o2:t2;i:integer;  begin  clrscr;  For i:=1 to 3 do  p1(o[i],I,chr(64+i));{инициализация полей массива объектов}  o2.p3(2,'D','???'); {инициализация полей объекта o2}  For i:=1 to 3 do  begin  o[i].Init;{запуск конструкторов массива объектов}  o[i].pout {вывод полей всех объектов массива}  end;  o2.Init; {запуск конструктора o2}  o2.pout;{вывод полей объекта o2}  end.  Особенности этой программы:  1. В ней используется массив из трех объектов типа t1 и один объект типа t2;  2. У этих двух типов есть виртуальные методы, поэтому должны быть и конструкторы, причем объект типа t2 наследует конструктор от типа t1.  Не запустив конструктор конкретного объекта, нельзя обратиться к его виртуальному методу вывода полей данных;  3. У внешней процедуры инициализации полей в объектах типа t1 сам тип объекта задается в качестве формального параметра **ob** , а при обращении к этой процедуре на месте первого параметра указывается фактическое имя инициализируемого объекта. Следовательно , переменные объектного типа ничем не отличаются от других типов с точки зрения их использования в качестве формальных и фактических параметров в процедурах и функциях.  При работе этой программы на экран выводятся строки:  t1.pout a1=1 a2=A  t1.pout a1=2 a2=B  t1.pout a1=3 a2=C  t2.pout a1=2 a2=D c=??? |