Процедуры и функции

Общие понятия

При разработке сложных программ нередко требуется один и тот же фрагмент кода использовать многократно в разных частях программы.

Часто эти фрагменты целесообразно поручить разным разработчикам.

Для того чтобы разделить работу, выделив какие-то подзадачи, удобно использовать аппарат процедур и функций.

Функция (процедура) это подпрограмма, т. е. последовательность инструкций, имеющая имя.

Процесс перехода к инструкциям функции (или процедуры) называется вызовом функции или обращением к функции. Процесс перехода от инструкций функции к инструкциям программы, вызвавшей функцию, называется возвратом из функции.

(во многих языках программирования, например C или Jawa различия между функцией и процедурой нет)

Объявление функции

```
function Имя (пар1 : тип1, ..., парК : типК) : Тип;
 var
{здесь объявления локальных переменных, этот раздел может
   отсутствовать}
begin
    {здесь инструкции функции}
    Имя := Выражение; {это результат}
end;
```

Пример

Объявление квадратичной функции с целыми коэффициентами (назовем ее **f**):

```
function f (a,b,c : integer; x : real) : real;
    {локальные переменные не требуются, var опускаем}
begin
     f := a*x*x+b*x+c; {это результат}
end;
```

Вызов функции

`Имя_функции (параметры)`

Вызов функции может находиться в любом месте программы (в выражениях, инструкциях, других функциях и т.п.)

Пример

Пусть нам нужно вычислить площадь под параболой $y = 4 - x^2$.

Тогда вызов функции в программе вычисления площади методом прямоугольников будет выглядеть так:

```
s:=s+f(-1,0,4,x);
```

Объявление процедуры

Процедура (а не функция) используется в двух случаях:

когда подпрограмма не возвращает в основную программу никаких данных (например, вычерчивает график в диалоговом окне);

когда подпрограмма возвращает в вызвавшую ее программу больше чем одно значение (например, решает квадратное уравнение и должна вернуть в вызвавшую ее программу два числа – корни уравнения)

В общем виде объявление процедуры выглядит так:

```
procedure Имя (    пар1: тип1; ... var парК: типК) ;
 var
{здесь объявление локальных переменных}
begin
{здесь инструкции процедуры}
end;
```

Вызов процедуры

Имя_процедуры (параметры);
Вызов процедуры является отдельной командой

Пример: решение квадратного уравнения с целыми коэффициентами
Объявление процедуры:

```
procedure kv_ur (a,b,c:integer;var x1,x2: real);
 var d:real;
begin
d:=b*b-4*a*c;
x1:=(-b-sqrt(d))/(2*a); x2:=(-b+sqrt(d))/(2*a);
end;
```

Вызов процедуры для решения уравнения $4-x^2=0$:

```
kv_ur(-1,0,4,k1,k2);
```

Стандартные (встроенные) функции:

sin, cos, StrToInt (и все аналогичные), EncodeDate, EncodeTime

Стандартные (встроенные) процедуры:

DecodeDate, DecodeTime

Место описания процедур и функций в программе зависит от того, используют ли они компоненты Delphi: если нет, то они помещаются в раздел implementation перед строкой {$R *.dfm}, а если используют, то после нее.

```
implementation
function f(x:real):real;
begin
   f:=x*x;
end;

{$R *.dfm}
procedure TForm1.p(x,y:real);
begin
   label2.caption:=floatToStr(x);
   label3.caption:=floatToStr(x+y);
end;
```

Запуск

Конструкторы и деструкторы.
Динамическое создание компонентов

Конструкторы и деструкторы

Каждому классу объектов присущи методы создания объекта (***конструктор***) и уничтожения его для освобождения занимаемой памяти (***деструктор***).

Для любого компонента, попавшего при визуальном проектировании в приложение из Палитры компонентов, конструктор и деструктор вызываются автоматически, незримо для программиста.

Если же объекты необходимо создавать динамически (во время выполнения приложения), то здесь нужен явный вызов конструктора (**Create**) и деструктора (**Destroy**).

Динамическое создание компонентов

Если компонент должен появляться динамически (во время выполнения программы, а не до запуска), он должен быть предварительно объявлен в разделе var:

```
var
…
 plita: TShape;
```

Программа создания компонента:

```
    plita:=TShape.Create(self);
    plita.parent:=self;
    plita.Left:=…;
    plita.Top:=…;
    …
```

Создание массива компонентов

Объявление массива кнопок

```
var
...
 m_but : array[1..100] of TButton;
```

Создание массива кнопок:

```
for i:=1 to n do begin
     m_but[i]:=TButton.Create(self);
     {создается очередной элемент массива кнопок m_but}
     m_but[i].parent:=self;
    {он появляется на форме}
     m_but[i].Top:=i*25;
     m_but[i].Left:=30;
        {кнопки размещаются одна под другой с интервалом
     25 пикселей между верхними границами}
     m_but[i].Caption:=IntToStr(i);
     {надпись на каждой кнопке соответствует ее номеру}
 end;
```

Создание нового класса

Объявление нового класса

```
type
   TPerson = class
    fName: string; // имя
       fGr:integer; //группа
    constructor Create(name:string;gr:integer);
    function info:string;
   end;
```

Объявление методов класса TPerson

```
constructor TPerson.Create(name:string;gr:integer);
    begin
        fName := name;
        fGr:= gr;
    end;

function TPerson.Info:string;
   begin
       result := fName + ' гр.' + IntToStr(fGr);
   end;
```

Создание нового класса

Объявление массива объектов класса TPerson

```
var
…
  List: array[1..100] of TPerson;
```

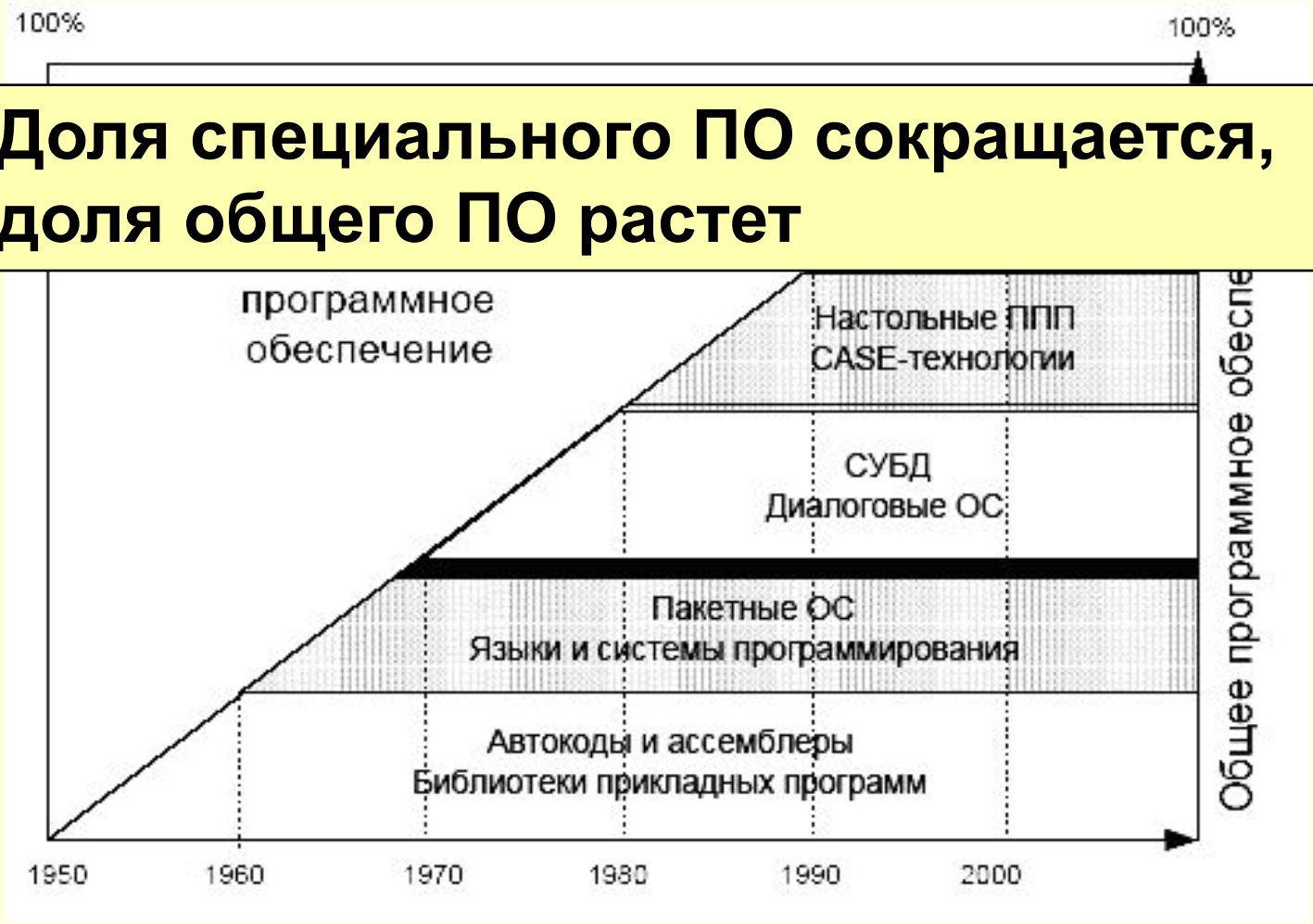
Последовательное создание массива объектов класса TPerson по кнопке

```
  List[n]:=TPerson.Create(Edit1.Text, SpinEdit1.Value);
  n:=n+1;
```

Запуск

Классификация и эволюция программного обеспечения

Классификация и эволюция программного обеспечения



Развитие языков программирования