Лекция 11.
Процедуры и функции ТР (Подпрограммы).

- Процедуры и функции - логически самостоятельные фрагменты программы, оформленные специальным образом и снабжённые именем.

- Обращение к подпрограмме (или ВЫЗОВ) происходит при упоминании имени ПП, за которым в скобках могут следовать параметры – список аргументов ПП.

  Результат выполнения операторов функции – некоторое вычисленное значение, которое присваивается имени функции.

- Процедура вызывается в отдельном операторе; функция встречается в правой части оператора присваивания как элемент выражения.

Для чего нужны ПП ?

1. Модульная структура программы.
2. Разбиение задачи на несколько задач, меньших по объёму.
3. Метод нисходящего проектирования.

Примеры вызова процедур:

Clrscr; Read(x,a,b); Close(fp); Delete(s, n, k)

Примеры вызова функций:

Y:= sin(x)+Abs(x-9) +2; z:= sqrt(cos(x)+3);

Описание ПП.

Процедура:

```
  Procedure ИМЯ_ПР ( список формальных параметров);
  <разделы описаний>
begin
    <раздел операторов процедуры>
end;
```

Функция:

```
Function ИМЯ_фун (список формальных параметров):
ТИП;
  <разделы описаний>
begin
    <раздел операторов функции>
end;
```

список форм. параметров может отсутствовать

Список формальных параметров процедуры.

Для каждого параметра задаются:
имя, тип и способ передачи.

Основные виды параметров:

1. Параметр – значение (имя : тип).
2. Параметр – переменная (var имя : тип).
3. Константы.
4. Нетипизированные параметры.

Пример:
Procedure Prima (n : integer; Var x : real; Const b : byte)

на место n можно подставлять выражение совместимого типа;

При вызове Prima передаётся значение, хранящееся по адресу n.

Локализация имён.

Глобальными называются переменные, которые описаны в главной программе. Время жизни глобальных переменных — с начала программы и до её завершения. Располагаются в сегменте данных.

В подпрограммах описываются *локальные* переменные. Они располагаются в сегменте стека, причем распределение памяти происходит в момент вызова подпрограммы, а ее освобождение — по завершении подпрограммы.

Локальные переменные автоматически не обнуляются.

Если локальное имя совпадает с глобальным, то действует локальное имя.

Список формальных параметров процедуры (продолжение).

При вызове процедуры на место параметра – переменной следует подставить фактический параметр в виде переменной того же типа, что и формальный параметр. Будет передана сама переменная (точнее, её адрес), а не её значение. Результат работы процедуры может передаваться только через параметры-переменные или через глобальные переменные.
На место параметра-переменной нельзя подставлять выражения.

Параметр-константа аналогичен параметру-значению, но обеспечивает более эффективный код. Нельзя изменять параметр константу в пределах процедуры.

ПРИМЕРЫ.

Что будет выведено на экран в результате выполнения программы

```
var a, b, c, d, e : word;

procedure Smile (a, b, c : word ; var d : word);
     var e: word;
     begin  c := a + b; d := 2*c; e := c div 2;
     writeln (‘c=‘, c, ‘ d=‘, d, ‘ e=‘, e );
end;
begin
     a :=3; b :=5;
     Smile (a, b, c, d);
     writeln (‘c=‘, c, ‘ d=‘, d, ‘ e=‘, e);
end.
```

Ответ:

c= 8 d=16 e=4

c= 0 d=16

e=0

Пример.

Заголовок процедуры имеет вид:

```
Procedure Prim(a:real; b:char; var c:real);
```

Переменные в вызывающей программе описаны так:

```
Var a : integer;  b, c : char;   d, x : real;
```

Какие из перечисленных ниже вызовов процедуры правильные?

```
1.   Prim(a,     b,   c);
2.   Prim(d+a, c,    x);
3.   Prim(x,     'c',   d);
4.   Prim(a,     b,    a+1);
```

1 – err

2 – right

3 – right

4 - err

Функции ТР (пример).

Заданы три числа a,b,c. Написать программу, которая проверяет, существует ли треугольник с такими сторонами, и если «да», то вычислить его площадь по формуле Герона.

1. Вводим числа a, b, c.
2. Если все они положительные и для любого из них выполняется неравенство треугольника : X< Y+Z, то можно вычислять площадь, иначе программа останавливается.
3. S=sqrt(p*(p-a)*(p-b)*(p-c)); где p=(a+b+c)/2.

Формула Герона.

```
program PRG_Geron;
 Var a,b,c,S,p :real;

 Function Tst(x,y,z:real):boolean;
 begin
 if(x>0)and (y>0) and (z>0) and (x<y+z)  and (y<x+z)
       and (z<x+y) then Tst := True else Tst := False;
 end;
Procedure Streug(x,y,z : real; Var St : real);
 var p : real;
 begin
 p:=(x+y+z)/2.;
 St:=sqrt(p*(p-x)*(p-y)*(p-z));
 end;
```

Формула Герона (продолжение).

```
begin
      write(' Введите стороны треугольника a,b,c=');
  readln(a,b,c);

  if (Tst(a,b,c)) then
  begin
              Streug(a,b,c,S);
writeln('Площадь треугольника =',S:10:5);
  end
                  else
 Writeln(' треугольника с такими сторонами не существует');
            Readln
          end.
```

Как передать в подпрограмму массив?

Нельзя написать так:

```
Procedure BD(a : array[1..10] of real)
```

Можно так:

```
Type amas = array [1..10] of real;
 ….
Procedure GD(a:amas);
```

Или так:

```
Const n=30
Type amas = array [1..n] of real;   ……..
Procedure GD(a:amas);
```

Можно передавать ОТКРЫТЫЕ МАССИВЫ
(без указания верхней границы массива)

```
Procedure OPN(axx : array of integer);
```

Функция HIGH(axx) возвращает максимальный индекс массива axx; минимальный = 0.

Передача массивов в процедуру.

Задача:
Вычислить векторное и скалярное произведение двух заданных векторов. Результат вывести на экран.

Передача массивов в процедуры (пример).

```
program Cross_Prod;

  Type vect=array[1..3] of real;
  Var u,v,w : vect;

  Procedure cross_p(a,b:vect; var c:vect);
  begin
  c[1]:= a[2]*b[3]-a[3]*b[2];
  c[2]:= a[3]*b[1]-a[1]*b[3];
  c[3]:= a[1]*b[2]-a[2]*b[1];
  end;
 Function dotp(a,b : vect) : real;
  Var i : byte; s : real;
  begin   s:=0;   for i:=1 to 3 do s:=s+a[ i ]*b[ i ];
  dotp:=s;
  end;
```

$$\begin{pmatrix} i & j & k \\ a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{pmatrix}$$

Передача массивов в процедуры (пример).

```
Procedure Vvod(Const s:char; var a:vect);
  begin
  write(' Input Vector  ', s ,' =');
  read(a[1], a[2], a[3]);
  end;

 Begin { основная программа}
  Vvod('U', u);
  Vvod('V', v);
  Cross_p(u,v,w);
  Writeln(' Cross product U x V = ',
                                  w[1]:8:3, w[2]:8:3, w[3]:8:3);
  Writeln('Dot product UV =', dotp(u,v) : 9 : 4);
  readln;
  end.
```

Выводы.

- При **вызове** подпрограммы после ее имени в скобках указываются аргументы, то есть те конкретные величины, которые передаются в подпрограмму
- Список аргументов как бы накладывается на список параметров и замещает их, поэтому **аргументы должны соответствовать параметрам по количеству, типу и порядку следования.**
- Для каждого параметра обычно задается его имя, тип и способ передачи.
- Либо тип, либо способ передачи могут не указываться.
- В заголовке подпрограммы нельзя вводить описание нового типа.

Выводы(продолжение).

- Для передачи в подпрограмму исходных данных используются параметры-значения и параметры-константы. Параметры составных типов (массивы, записи, строки) предпочтительнее передавать как константы.

Результаты работы процедуры следует передавать через параметры-переменные, результат вычисления функции — через ее имя.

Что будет выведено на экран?

```
var a : string;
Procedure U(a:char; var d:char);
   begin  a := ‘р’ ;  d:= ‘к’;
   end;
  begin
a:=‘потоп’;   U(a[1], a[5]); write (a) end.
```

```
var a,b,c,d : integer;
Procedure W(a : integer; var c : integer; var d: integer);
   begin  a := 5 ; c:= 7;   b:= 13;
   end;
begin a := 1 ; b:= 2;   c:= 0;
  W(b,a,c); write (a, ‘|’, b, ‘|’, c)        end.
```

Лекция 12. Модули в TP

- Структура модуля;
- Стандартные модули;
- Модуль CRT;

Модули в TP.

- Модуль – автономно компилируемая программная единица, включающая в себя различные компоненты раздела описаний – Type, Const, Var, Label, Procedure и Function и, возможно, некоторые исполняемые операторы инициирующей части.

Для чего нужны модули?

- Модуль как библиотека (может использоваться разными программами).
- Для разбиения сложной программы на обособленные части.
- Модули помогают преодолеть ограничение в один сегмент на объём кода исполняемой программы: код каждого модуля размещается в отдельном сегменте.

Основные правила для работы с модулями

- Имя модуля должно совпадать с именем файла, в котором он хранится (расширение **.pas)**
- Для подключения модуля к программе его надо предварительно откомпилировать: получить файл с заданным именем и расширением **.tpu** и сохранить его на диске.
- Имена подключаемых модулей перечисляются в директиве USES:

  Uses Crt, Graph, … и т.д.

Модули делятся на стандартные (входят в состав системы программирования) и пользовательские.

Структура модуля

```
unit имя;                    { заголовок модуля }

interface
    { интерфейсная секция модуля }
    { описание глобальных элементов модуля    (видимых извне) }

implementation
      { секция реализации модуля }
      { описание локальных (внутренних) элементов
       модуля }

begin
         { секция инициализации .  может отсутствовать }
end.
```

Структура модуля (продолжение)

- *В интерфейсной секции модуля* определяют константы, типы данных, переменные, а также заголовки процедур и функций.
- *В секции реализации* описываются подпрограммы, заголовки которых приведены в интерфейсной части. Кроме того, в этой секции можно определять константы, типы данных, переменные и внутренние подпрограммы.
- *Секция инициализации* предназначена для присваивания начальных значений переменным, которые используются в модуле.

Комплексные числа

$$z = x + iy; \quad x = \mathrm{Re}(z); \quad y = \mathrm{Im}(z); \quad i^2 = -1;$$

$$z_1 = x_1 + iy_1; \quad z_2 = x_2 + iy_2;$$

$$z_1 + z_1 = (x_1 + x_2) + i(y_1 + y_2);$$

$$z_1 * z_2 = (x_1 x_2 - y_1 y_2) + i(x_1 y_2 + y_1 x_2);$$

$$\frac{z_1}{z_2} = \frac{x_1 + iy_1}{x_2 + iy_2} = \frac{(x_1 + iy_1) * (x_2 - iy_2)}{(x_2 + iy_2) * (x_2 - iy_2)} =$$

$$= \frac{(x_1 x_2 + y_1 y_2) + i(-x_1 y_2 + y_1 x_2)}{(x_2)^2 + (y_2)^2};$$

Пример.

```
Unit A001;
Interface
type complex = record  Re, Im : real  end;
Procedure AddC( x,y : complex; var z : complex);
Procedure MultC(x,y : complex; var z : complex);
Implementation
Procedure AddC ( x,y : complex; var z : complex);
begin
z.re:= x.re + y.re;
z.im:= x.im + y.im;
end;
Procedure MultC;
begin
z.re:= x.re*y.re – x.im*y.im;
z.im:= (x.re*y.im + x.im*y.re) end;
end.
```

Стандартные модули TP.

- SYSTEM – подключается автоматически.
- DOS – доступ к возможностям ОС (файлы, время …)
- CRT – для работы в текстовом режиме
- GRAPH – для работы в графическом режиме
- PRINTER – для вывода на принтер. Исп. редко.
- OVERLAY – для загрузки программы по частям.
- STRINGS - для работы со строками, заканчивающимися нуль-символом.

WinDos – аналог модуля Dos

Модуль CRT.

- Обеспечивает эффективную работу с экраном, клавиатурой и динамиком в текстовом режиме.

Основные процедуры и функции:

1) KeyPressed : Boolean = False, если буфер клавиатуры пуст
   True, если БК содержит X1 символ.

2) ReadKey : Char = возвращает символ из БК.
   Если БК пуст, программа будет ждать нажатия клавиши.
   Функциональные клавиши заносятся в БК в виде #0#n

Примеры

Как очистить БК?

```
Program Exmp01;
Uses Crt;
Var s : char;
begin
While Keypressed  Do    s:=ReadKey;
end.
```

Определить расширенный код клавиши

```
Program Exmp02;
Uses Crt;
Var s : char ;
begin  repeat  s := ReadKey;
if ( s<>#0) then writeln ( ord(s)) else
            writeln (‘0’,ord (readKey) : 8) until s=#27 ;
end.
```

Основные процедуры и функции CRT:

3) TextMode(mode:word) – задаёт текстовый режим
В качестве Mode можно задать
BW40 = 0; (чёрно-белый 40x25)
Co40 = 1; (цветной 40x25)
BW80 = 2; (чёрно-белый 80x25)
Co80 = 3; (цветной 80x25)
При вызове TextMode сбрасываются все ранее сделанные установки цвета окон, экран очищается.

4) TextColor(c : byte) – задаёт цвет выводимых символов

5) TextBackGround(c : byte) – цвет фона

6) ClrScr – экран заполняется цветом фона;

Таблица const, задающих цвет:

Black	0 (черный)
Blue	1 (синий)
Green	2 (зеленый)
Cyan	3 (голубой)
Red	4 (красный)
Magenta	5 (малиновый)
Brown	6 (коричневый)
LightGray	7 (светло-серый)

DarkGray	8 (темно-серый)
LightBlue	9 (светло-синий)
LightGreen	10 (светло-зеленый)
LightCyan	11 (светло-голубой)
LightRed	12 (розовый)
LightMagenta	13 (светло-малиновый)
Yellow	14 (желтый)
White	15 (белый)
Blink	128 (мерцание)

white+Blink – мерцающий белый.

Пример. Цветные символы.

```
Program E_col;
Uses CRT ;
Var k : byte;
 Const  Col : array [1..15] of string[16] =('синий',  'зелёный',
'бирюзовый', 'красный', 'малиновый','коричневый',
'светлосерый', 'тёмносерый', 'светлосиний', 'св.зелёный',
'св.голубой', 'розовый', 'св.малиновый', 'жёлтый', 'белый');
begin
for  k := 1 to 15 Do begin  TextColor(k);
writeln ('цвет  номер ', k , '  =  ', Col [k] );
                              end;
TextColor (white+Blink);
Writeln('  мерцание текста');
ReadKey;
End.
```

Основные процедуры и функции CRT:

Window (x1, y1, x2, y2 : byte) – задаёт область экрана для вывода текста. Окно заполняется цв. фона.

GotoXY(x, y : byte) – перевод курсора в позицию (x,y)

WhereX , WhereY – текущие координаты курсора

Звук :	Sound(F: word),	F – частота в Гц
	Delay(t : word),	t - задержка выполнения программы в мс
	NoSound	отключить звук

Пример. Движущийся прямоугольник

```
Program MoveBar;
Uses CRT; var x,y,i : byte;
begin
TextBackGround(2);
ClrScr; x:=5; y:=10; Delay(1000);
for i := 1 to 30 do begin
x:=x+2; TextBackGround(9);
Window(x, y, x+10, y+5); ClrScr;
Delay(500);
TextBackGround(2);
ClrScr;
Delay(500);
end;
end.
```

Пример. Вложенные прямоугольники.

```
        Program CompBar;
Uses CRT; var k : byte;
begin
TextMode(Co80);
TextBackGround(2);
ClrScr; x:=5; y:=10; Delay(1000);
for k := 1 to 11 do begin
 TextBackGround(1+Random(15));
Window(2*k , k, 80-2*k, 26-k);
 ClrScr; Delay(1500);
end;
ReadIn;
TextBackGround(2);
ClrScr;
end.
```

Лекция 13. Модуль Graph.

- Назначение Graph;
- Подключение Graph к основной программе (графические режимы, процедура InitGraph);
- Основные процедуры и функции для работы в графическом режиме;
- Примеры программ;

Модуль Graph.

- Модуль обеспечивает работу с экраном в графическом режиме. Экран представляется в виде совокупности точек, или пикселей (= pixel =picture element).

Для определения положения пиксела вводится система координат : её начало – в левом верхнем углу и имеет координаты (0 ; 0).
Количество точек по осям (=РАЗРЕШЕНИЕ ЭКРАНА)
и доступные цвета определяются графическим режимом, который устанавливается спец. программой – графическим драйвером.

Что обеспечивает модуль Graph?

- Вывод линий и геометрических фигур заданным цветом и стилем;
- Закрашивание областей заданным цветом и шаблоном;
- Вывод текста выбранным шрифтом, заданного размера и направления;
- Задание «Окон» и отсечение по их границе;
- Работа с графическими страницами;

Порядок действий при работе с модулем GRAPH

- Подключить модуль: Uses Graph;
- Перевести экран в графический режим: процедура InitGraph;
- Установить параметры изображения;
- Вывести изображение;
- Вернуться в текстовый режим (если надо).

Графические драйверы и режимы.

- Драйвер обеспечивает взаимодействие программы с графическим устройством (монитором). Имеют расширение «bgi»

  ***.BGI**

- Графическе режимы: CGA, EGA, MCGA, режим VGA это:

  разрешение 640 x 480 и 16 цветов;

- Имя соответствующего драйвера:

  EGAVGA.BGI

Процедура InitGraph

- InitGraph(var Driver, Mode : integer; Path : String);

Driver – определяет тип графического драйвера;

Mode – задаёт режим работы графического адаптера.

Path – содержит путь к каталогу, содержащему файлы графических драйверов.

Можно (и нужно !) определять тип драйвера автоматически.

Это делается так:

```
Driver := detect;
InitGraph(Driver, Mode, 'd:\TP7\ALLBGI');
```

Процедуры для работы с графикой

- **GraphResult** – содержит код ошибки при выполнении графической операци; (grOk=0)
- **CloseGraph** – завешает работу в гр. ржиме;
- **RestoreCrtMode** – временное восстановление текст. режима;
- GetGraphMode; SetGraphMode; DetectGraph; и т.д.

Координаты, окна, страницы.

GetMaxX и GetMaxY – возвращают максим. координаты экрана;

GetX, GetY – координаты курсора;

ViewPort(x1,y1,x2,y2: integer; clipon : boolean);

MoveTo(x,y : Integer) – перемещает курсор в позицию (x,y)

MoveRel(dx,dy: integer) – относительное перемещение;

ClearDevice – заполняет экран цветом фона;

ClearViewPort – очищает графическое окно.

Линии и точки. (продолжение)

SetColor(c : word) – устанавливает цвет линий и символов;

SetBkColor(c: word) – цвет фона;

GetColor - возвращает текущий цвет;

GetMaxColor – максимальное значение кода цвета

Линии и точки.

PutPixel(x,y : integer; Color : word) – рисуем точку (x,y) цветом Color;

GetPixel(x,y:integer) : word - возвращает цвет пиксела;

Line (x1,y1,x2,y2) – рисуем линию текущ. цв. и стилем;

LineTo(x,y) – линия из тек. положения в (x,y);

LineRel(dx,dy) – линия из т.п. (a,в) в точку (a+dx, b+dy)

SetLineStile(Type,Pattern,Thick);
Type (тип линии)= { SolidLn, DottedLn, CenterLn,DashedLn, UserBitLn} = {0,1,2,3,4}
Pattern – задаётся только в случае UserBitLn;
Thick = {NormWidth ; ThickWidth}

Примеры констант модуля Graph

Константы шрифтов

Константа	Значение
DefaultFont	0 (растровый шрифт)
TriplexFont	1 (векторный шрифт)
SmallFont	2
SanSerifFont	3
GothicFont	4
HorizDir	0 (слева направо)
VertDir	1 (сверху вниз)

Примеры констант модуля Graph (продолжение)

Константы образцов закрашивания

Константа	Значение	Описание
EmptyFill	0	Закрашивание области фоновым цветом
SolidFill	1	Непрерывное закрашивание области
LineFill	2	Закрашивание ----------
LtSlashFill	3	Закрашивание /////
SlashFill	4	Закрашивание жирными линиями ////
и т.д.	5 .. 11, 12	\\\ \\\ +++ xxx прямоуг . . . …

Примеры констант модуля Graph (продолжение 2)

Константы стиля линии

Константа	Значение
SolidLn	0 (непрерывная)
DottedLn	1 (линия из точек)
CenterLn	2 (шрих-пунктир)
DashedLn	3 (пунктир)
NormWidth	1 (обычная толщина)
ThickWidth	3 (жирная линия)

Фигуры (Многоугольники, окружности и т.п.)

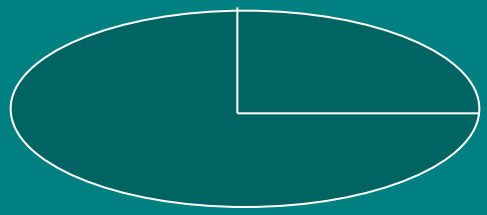
Rectangle(x1,y1,x2,y2) –

DrowPoly(N: word; var Points) – N –количество точек;
Points – массив точек,
каждая Type PointType = record x,y : word end;

Circle(x,y: integer; R: word) –

Arc(x,y,As,Ae,R) – дуга окружности: As и Ae – нач. и конечный углы в градусах, R – радиус.

Ellips(x,y,As,Ae,Rx,Ry)



Фигуры (Многоугольники, окружности и т.п.)

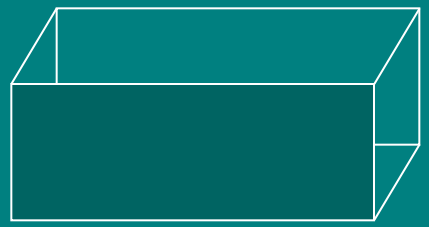
Bar(x1,y1,x2,y2) – закрашивает прямоугольник текущим образцом узора и цветом.
SetFill Style(Pattern,Color) – см таблицу.

Пример (фрагмент программы)

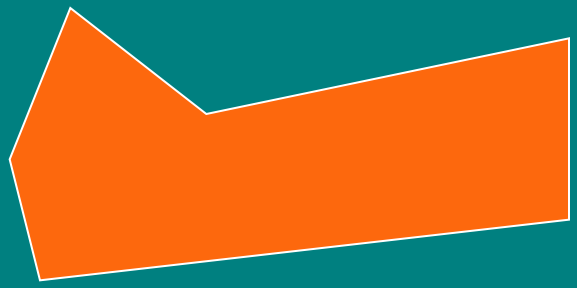
```
…..
 X:=GetMaxX div 6; Y := GetMaxY div 5;
For j :=1 to 2 do
   for k := 0 to 3 do    begin
Rectangle( (k+1)*x, (j+1)*y, (k+2)*x, (j+2)*y);
SetFillStyle( k+j*4,  j+1) ;
Bar ((k+1)*x+1, (j+1)*y+1, (k+2)*x-1, (j+2)*y-1))
                                end;
```

Фигуры (Многоугольники, окружности и т.п.)

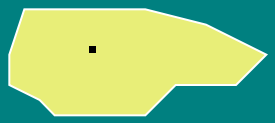
Bar3D (x1, y1, x2, y2, Depth: integer; Top : Boolean);



FillPoly(N : Word; var Coords) - закрашивает замкнутый многоугольник;



FloodFill(x ,y, BorderColor) – заполняет любую замкнут. фигуру



Вывод текста.

- OutTextXY(x,y,String)
- OutText(String)
- SetTextStyle(Font, Direct, Size) –

  Font – номер шрифта; (0 .. 10)

  Direct – код направления; (0 или 1)

  Size – размер шрифта; (1 .. 10)

размер

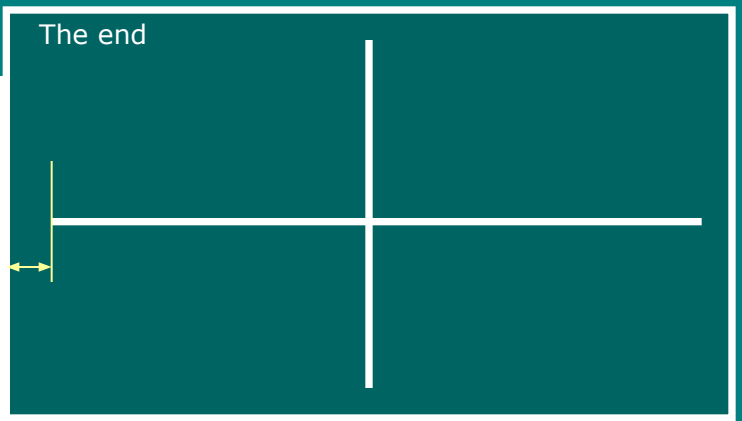
размер

Заливаем фигуру. Пример (фрагмент).

```
 ….
x := GetmaxX div 4;    y := GetMaxY div 4;
Rectangle (x, y, 3*x, 3*y) ;
SetViewPort (x+1,y+1, 3*x-1, 3*y-1,True);
SetFillStyle ( LtSlashFill, GetMaxColor);
Rectangle( 0, 0, 8, 20);
FloodFill (1,1, GetMaxColor);
OutTextXY(10, 25, ‘ Press Enter ..’);
Readkey;
Repeat  Randomize;
SetFillStyle( Random(12), Random(GetMaxColor));
x := Random (GetMaxX div 2); Y := Random (GetMaxY div 2);
c := Random (succ(GetMaxColor));  SetColor(c);
Circle( x, y,  Random (GetMaxY div 5));  FloodFill(x,y,c);
until KeyPressed;  …..
```

Пример

pole



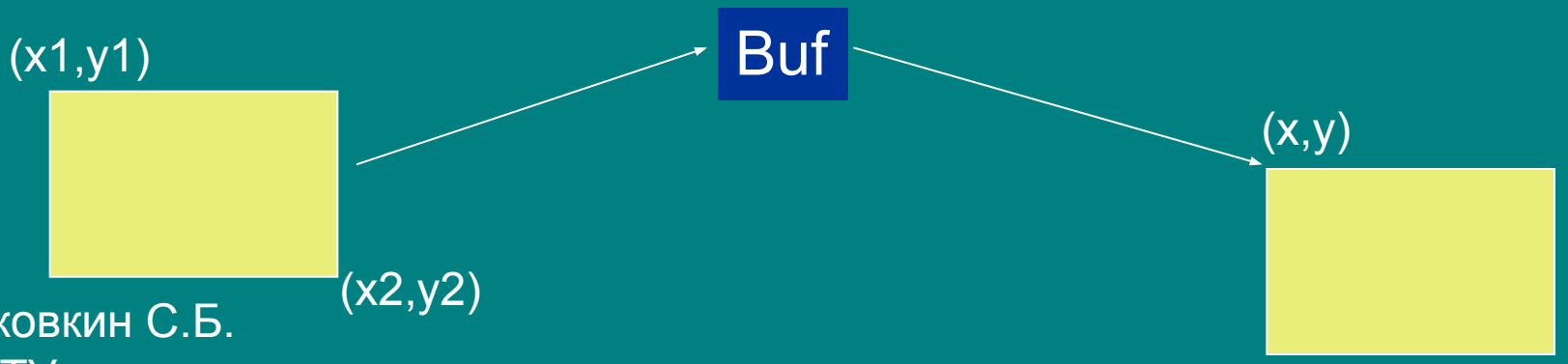

```
Program Grafika;
uses Graph;
const grDriver : integer = Detect;
        pole = 20;
var    grMode        : integer;
        maxX, maxY : integer;
begin
     { --------------------------инициализация графики ----- }
     InitGraph(grDriver, grMode, 'd:\tp\bgi');
     if GraphResult <> GrOK then begin
    writeln('Ошибка инициализации графики: ', GraphErrorMsg(GraphResult));
•              Halt  end;
•      maxX := GetMaxX; maxY := GetMaxY; { - вывод линий ----- }
•      Line(pole, maxY div 2, maxX - pole, maxY div 2);
•      Line(maxX div 2, pole, maxX div 2, maxY - pole);
•      SetColor(Cyan);   { ---------------   вывод текста}
•      SetTextStyle(GothicFont, HorizDir, 4);
•      OuttextXY(pole, pole, 'The end'); readln;
•      CloseGraph
•    end.
```

Сохранение и выдача изображений (1).

ImageSize(x1, y1, x2, y2) - возвращает размер памяти в байтах для прямоугольной области.

GetImage(x1, y1, x2, y2 : integer; var Buf) – помещает в память копию прямоугольной области изображения; изображение храниться в переменной Buf.

PutImage (x, y, Buf, Mode) – Выводит в заданное место экрана то, что хранилось в Buf. Mode – задаёт способ вывода.



Сохранение и выдача изображений (2).

Режимы вывода изображения MODE:

NormalPut	0	Замена изображения на копию из Buf
XorPut	1	Исключающее «или»
OrPut	2	Обычное «или»
AndPut	3	Логическое «и»
NotPut	4	Инверсия изображения: Red = 4= 0100 -> 1011 = LCyan

Сохранение и выдача изображений (3).

Для выделения памяти под изображение используем динамическую память:

Пример (фрагмент программы):

```
var Buf : Pointer;
…

 Size := ImageSize(x1, y1, x2, y2) ;
 GetMem (Buf, Size) ;
 GetImage (x1, y1,  x2,  y2, Buf^) ;
 PutImage (x1,y1, Buf^, XorPut) ; {стёрли то, что было};
…..
{ или PutImage(x, y, Buf^, NormalPut);  в новом месте}
Delay(1000);
```

Графика в Delphi (1)

1) Используется свойство CANVAS объектов Forma или Image.
2) Canvas – объект типа TCanvas.
3) Методы этого типа позволяют выводить графические примитивы: точки, линии, окружности, прямоугольники т.п.)
4) Свойства Tcanvas: цвет, толщина, стиль линий; цвет и вид заливки областей; характеристика шрифта при выводе текста.
4) Canvas – холст, состоящий из отдельных пикселей, с координатами (x,y)
5) Forma1.Canvas.Rectangle(15,25,80,90) - прямоугольник

Графика в Delphi (2)

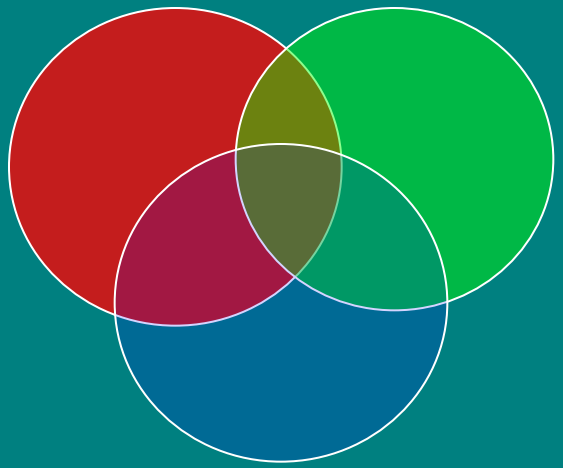
1) Размеры Canvas:
для Image: Canvas.Height и Canvas.Width
Для формы: ClientHeight и ClientWidth
2) Карандаш – Canvas.Pen (точки, линии прямоугольник, окруж.)
Кисть – Canvas.Brush
3) Свойства PEN: Color, Width, Style, Mode (режим отображения)
4) Canvas.Pen.Color:= clGreen;
Canvas.Pen.Width:=3;
5) Свойства Brush : Color, Style
6) Form1.Canvas.TextOut(x,y,Текст)
СвойстваЖ

Основные положения теории цвета

- Световой поток падает на сетчатку глаза от отражающего или излучающего объекта.
- Цветовые рецепторы делятся на три группы: (красный - зелёный - синий).
- Различают аддитивное и субтрактивное цветовоспроизвдение.

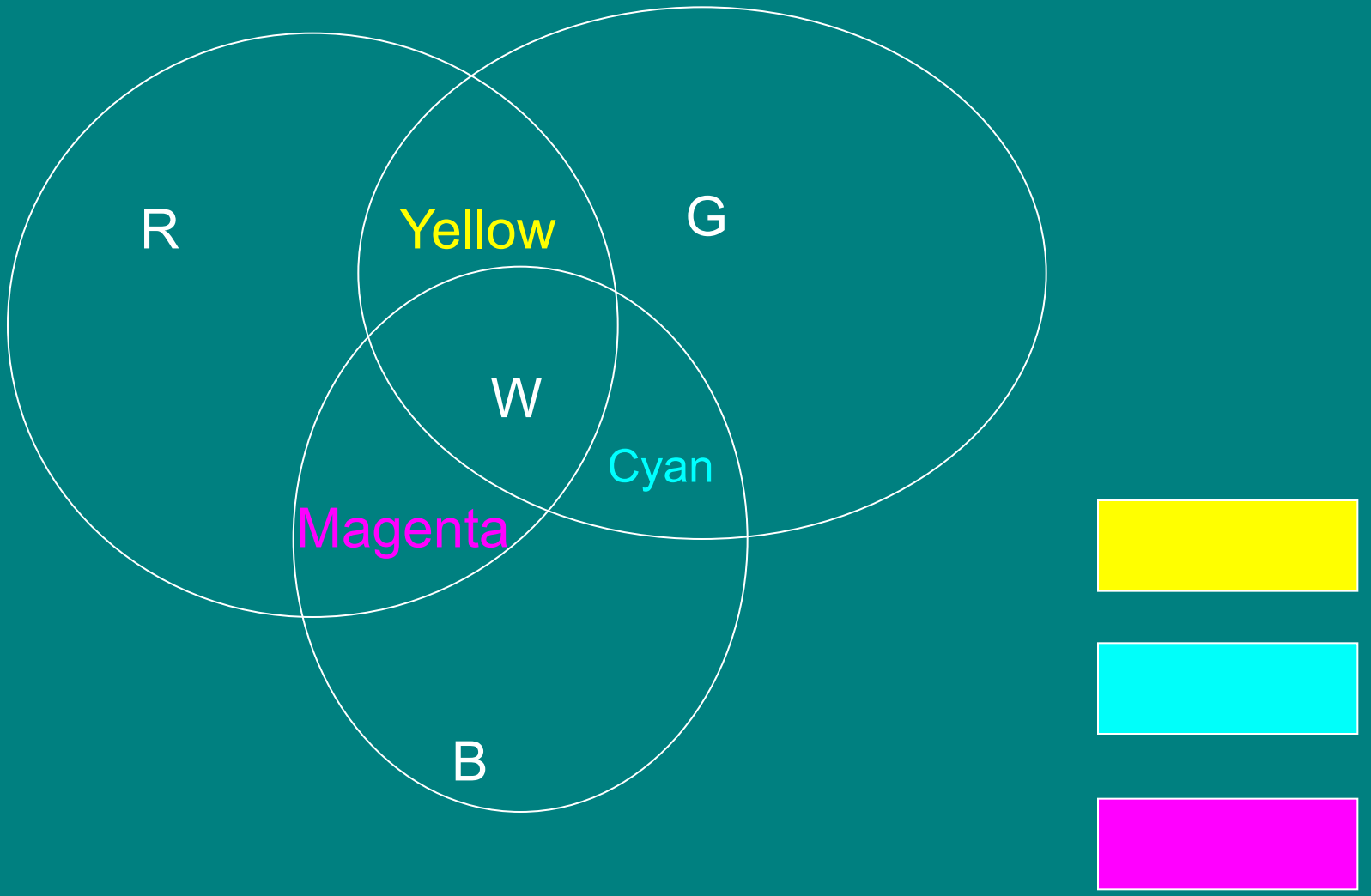
Модель RGB.

- W = R+G+B
- W – R = G+B = Cyan
- W – G = R+B = Magenta
- W – B = R+G = Yellow



Модель CMYK

Диаграмма цветов RGB

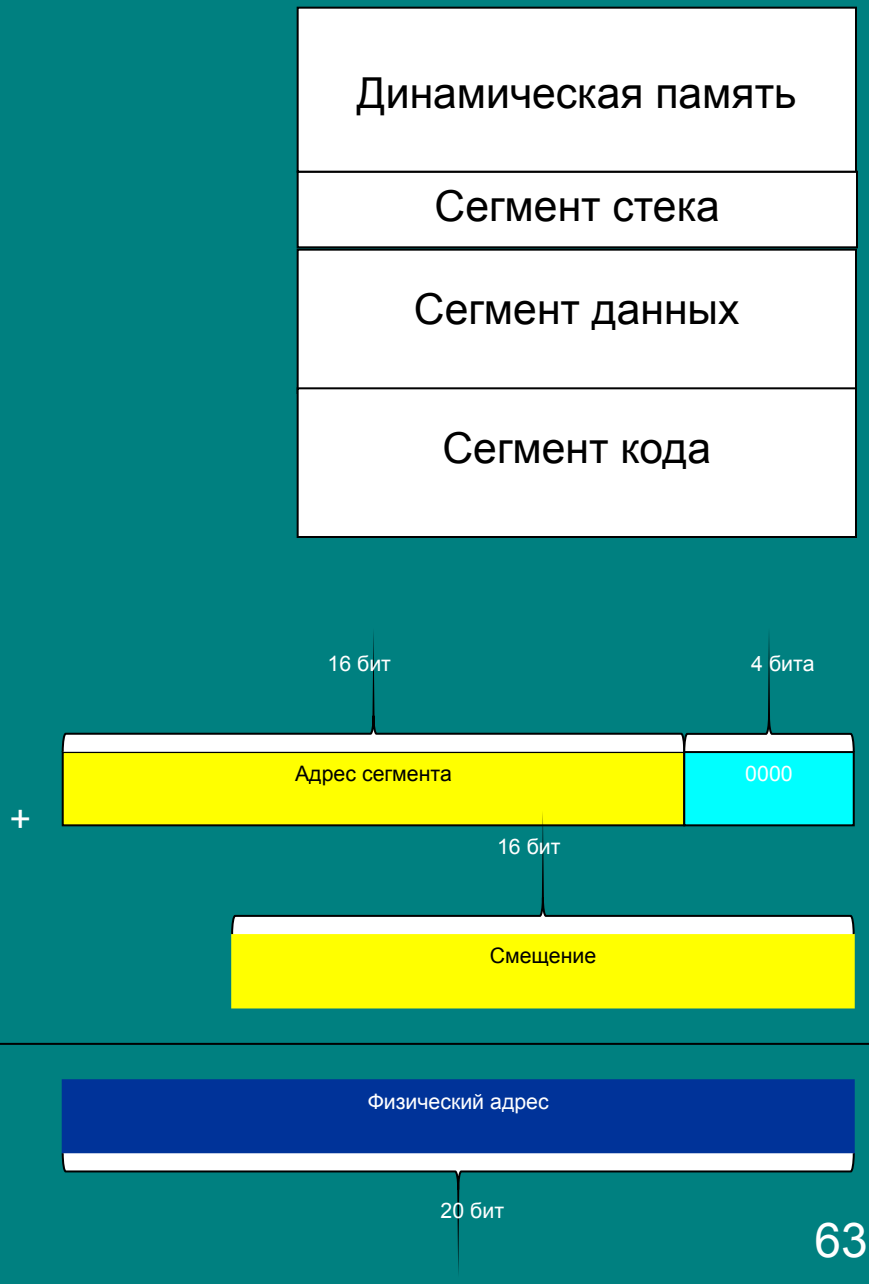


Лекция 14.
Указатели. Динамическая память.

Структура программы в оперативной памяти.

В IBM PC-совместимых компьютерах память условно разделена на сегменты. Адрес каждого байта составляется из номера сегмента и смещения.

Компилятор формирует **сегмент кода**, в котором хранится программа в виде машинных команд,
сегмент данных, в котором выделена память под глобальные переменные программы,
и **сегмент стека,** предназначенном для размещения локальных переменных подпрограмм.



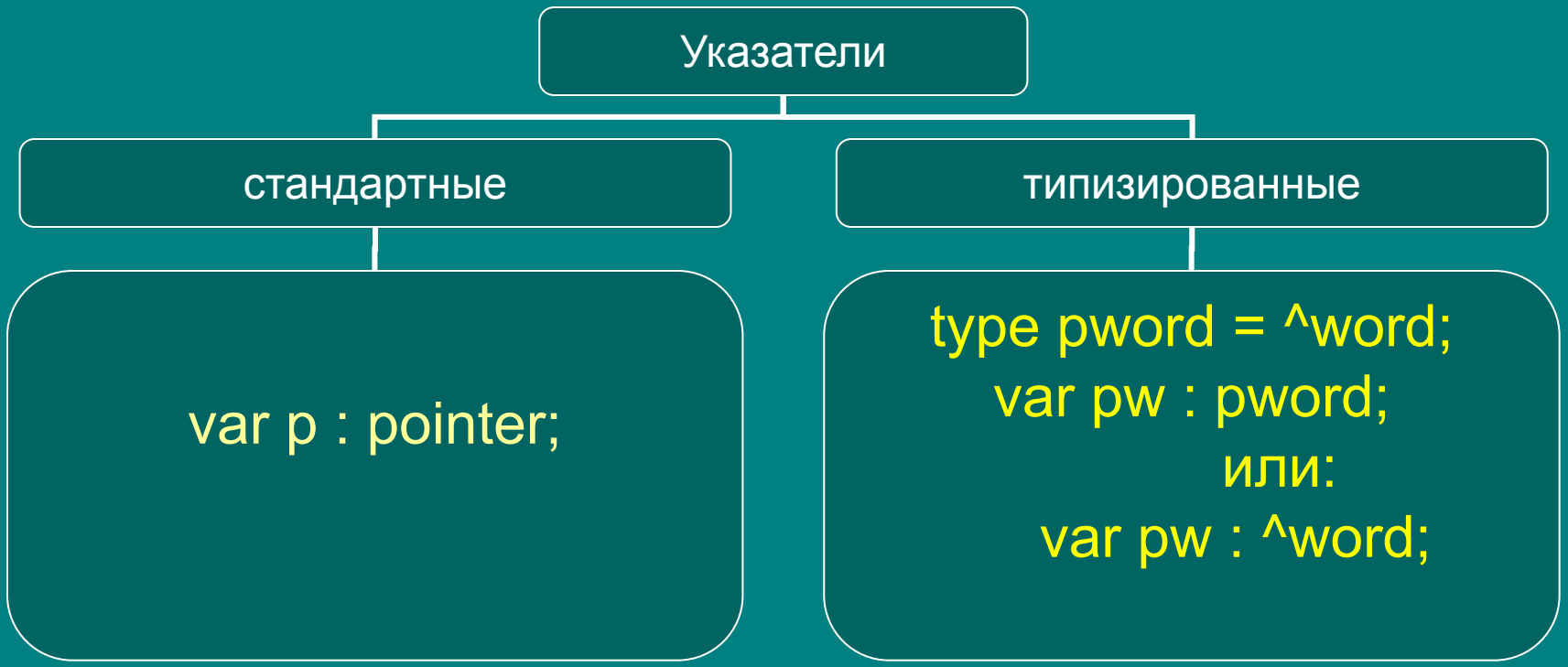
Основные понятия:

- Переменные, предназначенные для хранения адресов областей памяти, называются ***указателями***.
- В указателе можно хранить адрес данных или адрес программного кода.
- Адрес занимает четыре байта и хранится в виде двух слов, одно из которых определяет сегмент, второе — смещение.

Динамические переменные

- ДП создаются в динамической памяти (в хипе, в куче) во время выполнения программы.
- Обращение к ДП осуществляется через указатели.
- С помощью ДП можно обрабатывать данные, объём которых заранее не известен.

Виды указателей:



Операции с указателями:

- Для указателей определены операции:
  - присваивания;

    p1 := p2;
  - проверки на равенство и неравенство:

    if p1 = p2 then … или if p <> nil then …

Правила присваивания указателей

- Любому указателю можно присвоить стандартную константу **nil**, которая означает, что указатель **не ссылается** на какую-либо конкретную ячейку памяти:

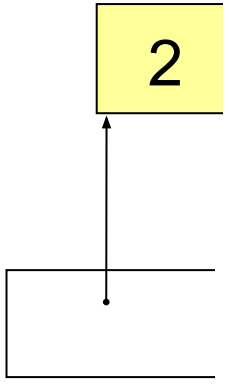
  p1 := nil;
- Указатели стандартного типа pointer совместимы с указателями любого типа.
- Указателю на конкретный тип данных можно присвоить только значение указателя того же или стандартного типа.

Операция разадресации

применяется для обращения к значению переменной, адрес которой хранится в указателе:

```
var p1: ^word;
…
p1^ := 2;     inc(p1^);    writeln(p1^);
```

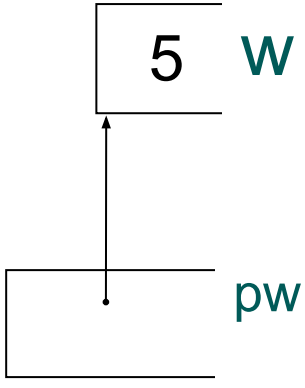
С величинами, адрес которых хранится в указателе, можно выполнять любые действия, допустимые для значений этого типа.



Операция @ и функция addr

позволяют получить адрес переменной:

```
var w   :   word;
    pw  : ^word;
...
pw := @w;
{ или pw := addr(w); }
pw^ := 5;
```



Стандартные функции для работы с указателями:

- seg(x) : word — возвращает адрес сегмента для x;
- ofs(x) : word — возвращает смещение для x;
- addr(x) : pointer – содержит адрес аргумента x;
- cseg : word — возвращает значение регистра сегмента кода CS;
- dseg : word — возвращает значение регистра сегмента данных DS;
- ptr(seg, ofs : word) : pointer — по заданному сегменту и смещению формирует адрес типа pointer.

Пример. Указатели.

```
program ka;
var w,a, i:integer;
    pw,pq:^integer;
    us,ut:word;
    begin
    write(‘ввод w, a='); readln(w,a);
pw:=@w;
    pq:=addr(a);
    us:=seg(w);
    ut:=ofs(w);
    writeln(us);
    writeln(ut);
  writeln(' w+a =',pw^+pq^);
  readln;
```

```
writeln(' a-w =',pq^ - pw^);
        readln;
  pw:=pq;
  writeln('pw^ -pq^ = ',pw^-pq^);
  new(pw);
  new(pq);
  pw^:=-25;
  pq^:=60;
  writeln('pw+pq=', pw^+pq^);
readln;
dispose(pw);
dispose(pq);
    end.
```

Динамические переменные.

создаются в хипе (Heap) во время выполнения программы с помощью подпрограмм **NEW** или **GetMem**:

- Процедура **new**(var p : тип_указателя)
- Функция **new**(тип_указателя) : pointer

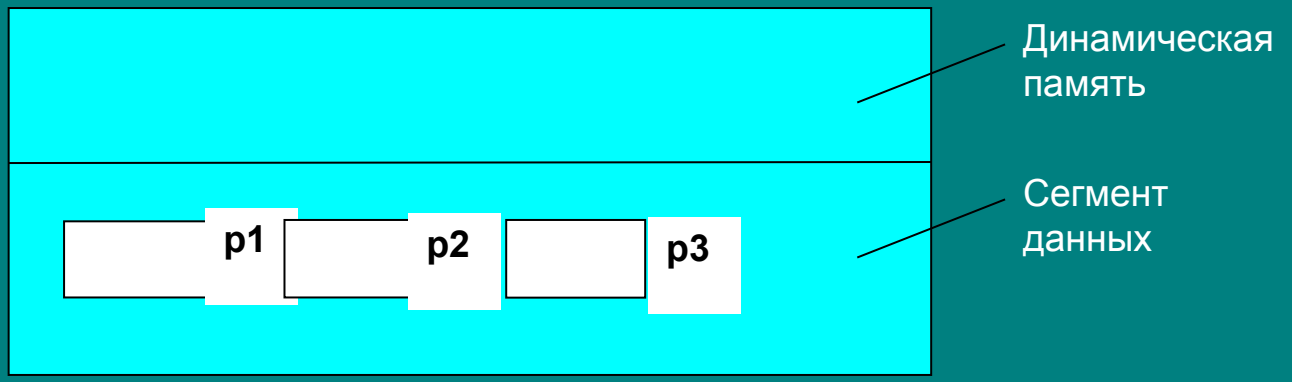
Процедура и функция **NEW** применяются только для типизированных указателей.

- Процедура **getmem**(var P : pointer; size : word) – выделяет в хипе участок в size байт; адрес его начала хранится в P.

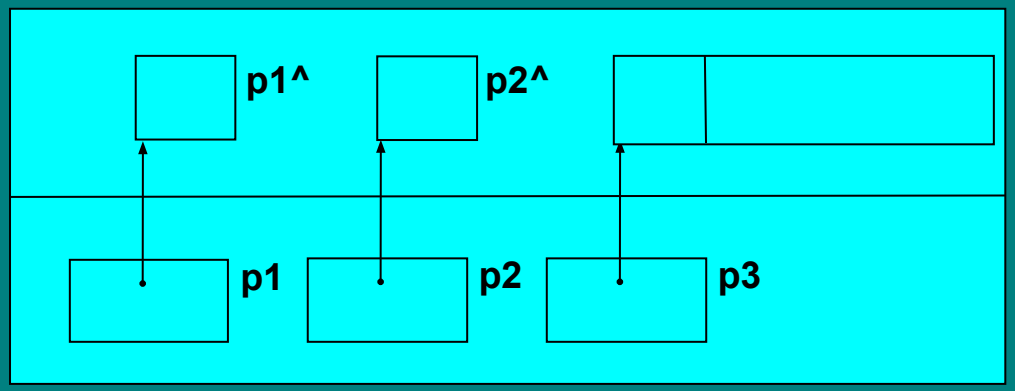
Эту процедуру можно применять и для указателей типа pointer.

Пример работы с динамическими переменными

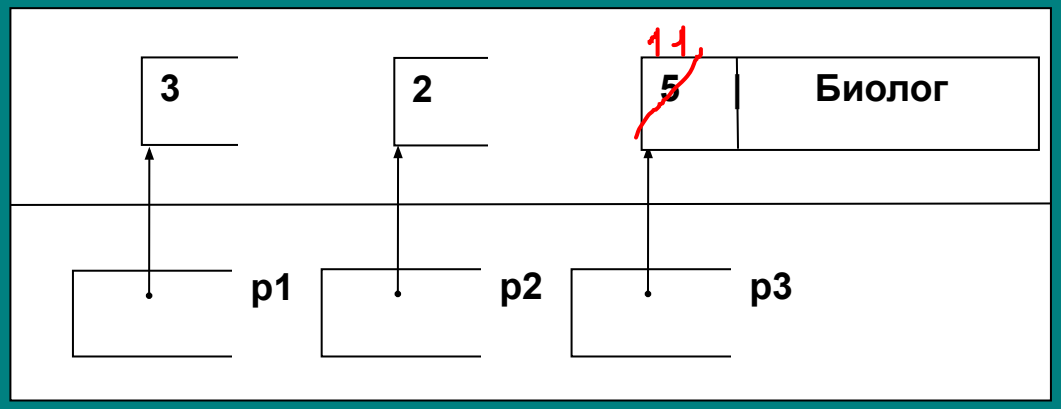
```
type rec = record
             d : word;
             s : string;
           end;
           pword = ^word;

var   p1, p2 : pword;
      p3     : ^rec;
```




```
...
new(p1);
p2 := new(pword);
new(p3);
```

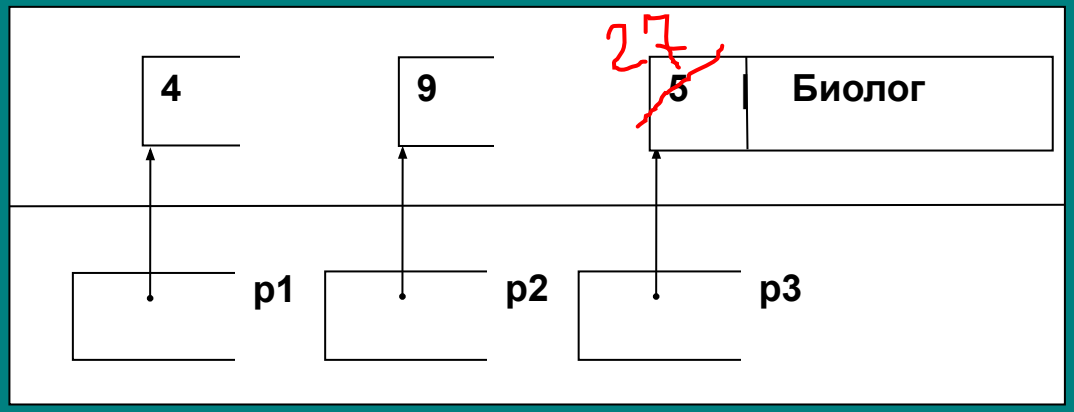


```
p1^ := 3;     p2^ := 2;

p3^.d := p1^+2;

p3^.s := ‘Биолог';
```



Динамические переменные можно использовать в операциях, допустимых для величин соответствующего типа:

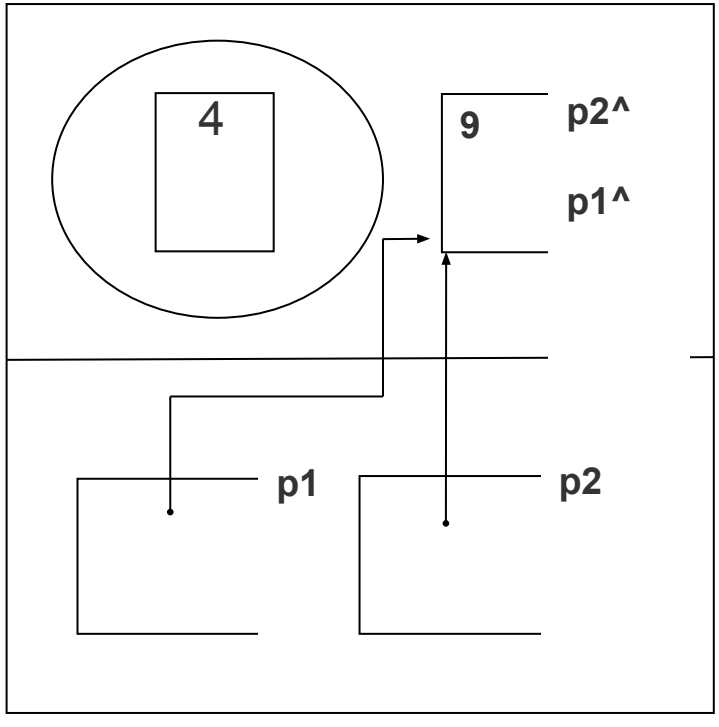
```
inc(p1^);

p2^ := p1^ + p3^.d;

with p3^ do writeln (d, s);
```



Мусор

При присваивании указателю другого значения старое значение теряется.

Это приводит к появлению мусора (обозначен овалом), когда доступа к участку динамической памяти нет, а сам он помечен как занятый.



Освобождение динамической памяти

- Процедура **Dispose(var p : pointer)**

освобождает участок памяти, выделенный процедурой New.

- Процедура **Freemem(var p : pointer; size : word)**

освобождает участок памяти размером size, начиная с адреса p.

Если память выделялась с помощью **New,** следует применять **Dispose**, в противном случае — Freemem.

- Значение указателя после вызова этих процедур становится неопределенным.

Лекция 15. Динамические структуры данных.

Динамические структуры данных.

ДСД – способ организации данных, при котором память распределяется во время работы программы по мере необходимости отдельными блоками, связь между которыми осуществляется с помощью указателей;
размещение данных происходит в динамической памяти.
В отличие от массивов и записей ДСД могут занимать несмежные участки памяти.

- Линейные списки
- Стеки
- Очереди
- Бинарные деревья

ДСД

Элемент любой ДСД состоит из двух частей:

1) информационной;
2) указателя;

Элемент ДС описывается в виде записи.

Пример:

```
type
pnode = ^node;
node = record
        d : word;
        s : string;
        p : pnode;
 end;
```

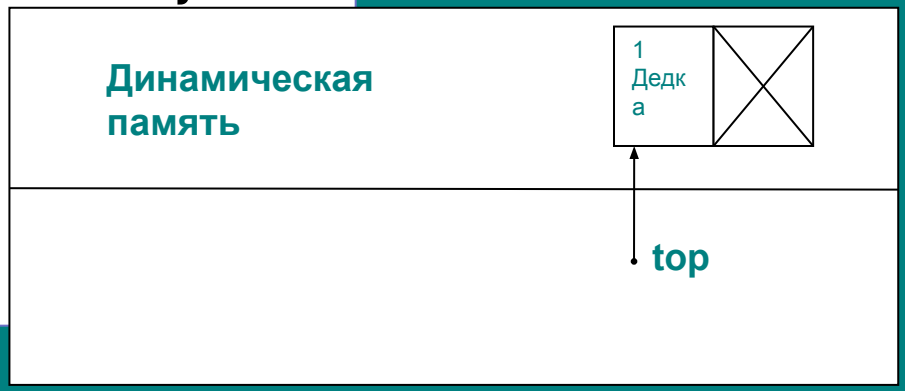
Стеки.

Принцип: LIFO = last in – first out.

Для работы со стеком используются две статические переменные:

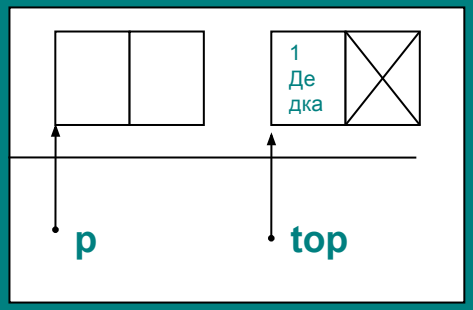
- указатель на вершину стека;
- вспомогательный указатель:

```
type   pnode = ^node;
       node = record  d : word;  s : string;
       p : pnode;  end;
var top, p : pnode;
Begin
{Создание первого элемента стека:}
new(top);
top^.d := 1;
top^.s := ‘Дедка';
top^.p := nil;
```

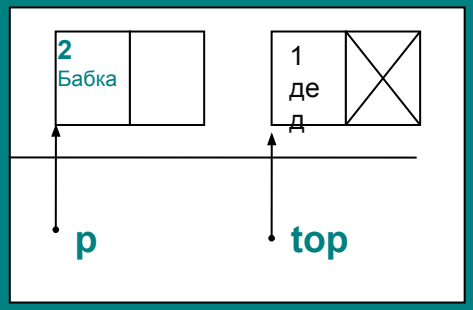


Добавление элемента в стек.

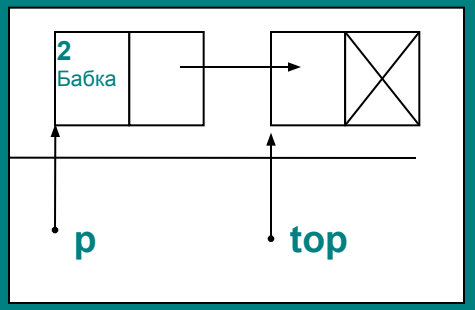
1. new(p);



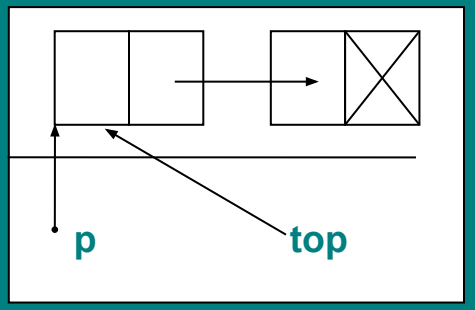
2. p^.d := 2;
   p^.s := ‘Бабка’;



3. p^.p := top;

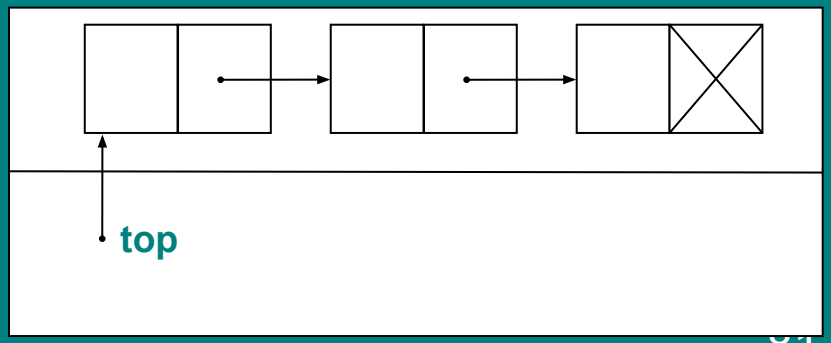


4. top := p;



Выборка из стека.

```
with  top^ do write(d,s);
p:=top;     top := top^.p ;
dispose(p);
```



Очередь

Принцип FIFO = first in – first out.
Новые элементы добавляются в один конец, а выборка - из другого конца очереди.
Применяется при буферизации ввода-вывода, диспетчеризации задач в ОС…
При работе с очередью используются указатели на её начало и на её конец, а также вспомогательный указатель.

Порядок действий:

- Начальное формирование очереди – создание первого элемента
- Добавление элемента в конец очереди;
- Выборка элемента

Формирование очереди

- Type pnode=^node;
- node=record
- d:word;
- s:string;
- p:pnode; end;
- Var beg, fin,p : pnode;
- ===== создаём первый элемент очереди ======
- New(beg) {выделяем динамическую память}
- beg^.d:=1; beg^.s:='первый'; beg^.p:=nil; fin:=beg;
- ========== добавляем элемент в очередь ======
- New(p);
- P^.d:=2; p^.s:='второй'; p^.p:=nil;
- Fin^.p=p; { предпоследний элемент указывает на последний}
- Fin:=p; { указатель на конец очереди }

Выборка из очереди

- Writeln(beg^.d, beg^.s);
- P:=beg;
- beg:=beg^.p
- dispose(p);
- If beg=nil then fin=nil;

Линейные списки

Каждый элемент ЛС связан со следующим и, возможно, предыдущим. Каждый элемент ЛС содержит ключ – это часть поля данных. Ключ идентифицирует элемент.

Можно выполнять следующие операции:

- начальное формирование списка;
- добавление элемента в конец списка;
- чтение элемента с заданным ключом;
- вставка элемента в заданное место списка;
- удаление элемента с заданным ключом;
- сортировка списка по ключу.

Линейные списки (2)

- Стек и очередь – частный случай линейного списка.
- При чтении элемент ЛС не удаляется.
- Для работы с ЛС надо определить указатель на его начало.

Работа со списком. Пример.

```
Program List;
type pe=^tpelem;
  tpelem=Record t:real; p:pe end;
     Var plm,beg: pe;
         x:real; ch:char; N:byte;

  Begin
         New(plm); beg:=plm;
         plm^.p:=plm;
   while plm^.p<>Nil do begin
   write('Введите t=');
readLn(plm^.t);
Write('Продолжить?(Y/N');
    readln( ch);
```

```
If (ch='y') or (ch='Y') then
begin New(plm^.p);
 plm:= plm^.p end
                        else
plm^.p:=nil
                        end;
    writeLn('Читаем список =');
    plm:=beg;   N:=1;
    repeat
    writeln(N, ':', plm^.t:8:3);
    plm:=plm^.p;
    Inc(N);
    until plm=Nil;
    end.
```

Линейные списки. Пример Не ГОТОВ (3)

```
Program LLL;
const  n=5;
type pnode = ^node;
        node = record  d : word; s : string; p : pnode;  end;
var beg : pnode;  i , key, option : word; s1 : string;
const text : array[1..n] of string =
                                (‘один’, ‘два’, ‘три’, ‘четыре’, ‘пять’);
procedure addL(var beg : pnode; d : word; s1 : string);
var p, t : pnode;
begin
New(p);  p^.d := d ; p^.s :=s1; p^.p := nil;
If beg = nil then beg := p
                else  begin   t := beg
```

Лекция 16. Объектно-ориентированное программирование.

Основные принципы ООП

- Инкапсуляция
- Наследование
- Полиморфизм

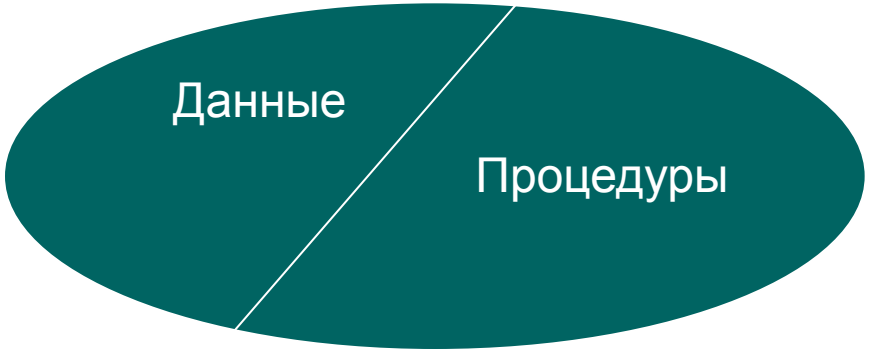
ООП. Инкапсуляция

Основная идея – связать в одно целое данные и подпрограммы для их обработки.

Объект – совокупность данных, характеризующих его состояние, и процедур (алгоритмов) их обработки.

Предметная область представляется в виде совокупности объектов.

ИНКАПСУЛЯЦИЯ



Данные = поля

Процедуры= методы

ООП. Наследование.

Важное значение имеет возможность многократного использования кода. Для объекта можно определить наследников, корректирующих или дополняющих его поведение.

Наследование - свойство объектов порождать потомков.
Потомок наследует все поля и методы родителя.
Можно дополнять существующие поля и методы.
Можно модифицировать методы объекта родителя.

Можно создавать иерархии объектов.
Объект может иметь только одного предка и несколько потомков.

ООП. Полиморфизм

- ООП позволяет писать гибкие, расширяемые и читабельные программы.
- Во многом это обеспечивается благодаря полиморфизму, под которым понимается возможность во время выполнения программы с помощью одного и того же имени выполнять разные действия или обращаться к объектам разного типа.
- Чаще всего понятие полиморфизма связывают с механизмом виртуальных методов.

Достоинства ООП

- использование при программировании понятий, близких к предметной области;
- возможность успешно управлять большими объемами исходного кода благодаря инкапсуляции, то есть скрытию деталей реализации объектов и упрощению структуры программы;
- возможность многократного использования кода за счет наследования;
- сравнительно простая возможность модификации программ;
- возможность создания и использования библиотек объектов.

Недостатки ООП

- некоторое снижение быстродействия программы, связанное с использованием виртуальных методов;
- идеи ООП не просты для понимания, в особенности для практического использования;
- для эффективного использования существующих объектно-ориентированных систем требуется большой объем первоначальных знаний;
- неграмотное применение ООП может привести к значительному ухудшению характеристик разрабатываемой программы.

Объект (класс)

- Объект – это тип данных; его определение находится в разделе описания типов;
- Объект похож на тип RECORD, но кроме полей данных в нём можно описывать методы – подпрограммы для работы с полями объекта.
- Поля и методы = элементы объекта;
- Внутри объекта описываются только заголовки методов.

Объект (продолжение)

- Видимостью элементов можно управлять директивами ***private*** и ***public***
- Количество разделов privat и public – произвольное.
- Всё, что расположено после private, является невидимым из внешних файлов.
- По умо все элементы являются public.

Классы и объекты

- Type Tmns=class
- **Private**
- x,y:real;
- s:string[15];
- **Public**
- **Constructor create;**
- **Procedure fp(u:word);**
- **Function ff(w:real):boolean;**
- **…..**
- **End;**

- **Могут быть и другие секции …**

- **Описание методов в разделе implementation.**

- Var exam: Tmns;
- Procedure Tmns.fp(u:word);
- Function Tmns.ff(w:real):boolean;

- Begin
- Exam:= Tmns.create;

Графика Delphi

- Свойство Canvas
- Методы типа Canvas (рисование точек, линий, окружностей, прямоугольников)
- Свойства (цвет, толщина, стиль линий, цвет и вид заливки областей, свойства шрифта)

- Image.Canvas
- Shape.Canvas

Pen & Brush

- Свойства Pen:
- Color, Width,
- Canvas.Pen.Width:=2;
- Canvas.Pen.Color:={clBlack, clRed …}
- Canvas.Brush.Color – цвет заполнения
- Canvas.Brush.Style – стиль заполнения