СИСТЕМА АНИМАЦИИ

OS	Animation -> Anim	Application	Render3D
WinMain: - регистрация класса окна - создание окна - цикл сообщений *while (GetMessage(...))* *DispatchMessage(...);* WinFunc: WM_GETMINMAXINFO: . . . WM_CREATE: WM_SIZE: WM_TIMER: WM_ERASEBKGND: WM_PAINT: WM_DESTROY: Input: WM_MOUSE*** WM_KEY*** WM_MOUSEWHEEL:	AnimInit(hWnd); *инициализация* AnimClose(); *деинициализация* AnimResize(W, H); *изменение размера кадра* AnimCopyFrame(hDC); *копирование кадра* AnimRender(); *построение кадра (опрос устройств ввода, обновление таймера, опрос всех элементов анимации и вызов у них функции Response, очистка кадра и вызов у всех функций анимации Render)*	массив элементов анимации UNIT: [Init(Anim); *инициализация* Close(Anim) *деинициализация* Response(Anim) *отклик элемента анимации на смену кадра (обработка клавиатуры, таймера и т.п.)* Render(Anim)] в каждую функцию элемента анимации приходит параметр - Anim - текущий контекст (параметры) анимации: - клавиатура - мышь - джойстик - таймер - параметры ввода (...) - параметры визуализации (hDC, ...)	VEC: DBL VEC MATR RndObjLoad() RndObjDraw()

Как сделать элементы анимации с разным поведением?

1.разное поведение -

указатели - pointers

данные:

```
int x, y, *p;

p = &x;
*p = 30;
p = &y;
*p = 59;
```

действия:

```
void f( int x )
{
  printf("%d", x * x);
}

void g( int x )
{
  printf("%d", x * x * x);
}

void main( void )
{
  int r = 1;
```

```
  void (*p)( int x ) = f;

  while (r)
  {
    p(rand());
    switch (getch())
    {
    case 27:
      r = 0;
      break;
    case '1':
      p = f;
      break;
    case '2':
      p = g;
      break;
    }
  }
}
```

пример:

```
typedef struct
{
  void (*Draw)( void );
} SHAPE;

/* sphere */
void SphDraw( void )
{
  printf("sphere");
}

/* cube */
void CubeDraw( void )
{
  printf("cube");
}
```

	void f(int n)
	{
	  int a[n];
	  . . .
	}
	
	void f(int n)
	{
	  int *a = malloc(sizeof(int) * n);
	  if (a == NULL)
	  return;
	  . . .
	  free(a);
	}

```
----------------------------------
SHAPE * CreateCube( void )
{
  SHAPE *S = malloc(sizeof(SHAPE));

  S->Draw = CubeDraw;
  return S;
}
SHAPE * CreateSph( void )
{
  SHAPE *S = malloc(sizeof(SHAPE));

  S->Draw = SphDraw;
  return S;
}
----------------------------------
SHAPE *Shapes[100];

void main( void )
{
  /* создание */
  Shapes[0] = CreateSph();
  Shapes[1] = CreateCube();
  Shapes[2] = CreateSph();

  /* рисование */
  for (i = 0; i < 3; i++)
    Shapes[i]->Draw();

  /* уничтожение */
  for (i = 0; i < 3; i++)
    free(Shapes[i]);
}


    Shapes[i]->Draw();   <=>   (*Shapes[i]).Draw();
```

Это основной принцип ОБЪЕКТНО-ОРИЕНТИРОВАННОГО ПРОГРАММИРОВАНИЯ:

ДАННЫЕ УПРАВЛЯЮТ СОБСТВЕННЫМ ПОВЕДЕНИЕМ

2.как сделать данные (элементы анимации) разного размера?

ООП:
- ИНКАПСУЛЯЦИЯ (incapsulation) - сокрытие данных
- НАСЛЕДОВАНИЕ (iheritance) - включение полей одних структур - в другие
- ПОЛИМОРФИЗМ (polymorph, override) - различное поведение наследуемых функций

решение проблемы:
в начале каждой структуры (в нашем случае - в начале каждого объекта анимации) - одинаковый набор данных (полей):

Создадим общий тип объекта анимации:
[каждая функция объекта будет получать указатель на сам объект и указатель на контекст анимации]

```
typedef struct tagUNIT UNIT;
struct tagUNIT
{
  VOID Init( UNIT *Uni, ANIM *Ani );
  VOID Close( UNIT *Uni, ANIM *Ani );
  VOID Response( UNIT *Uni, ANIM *Ani );
  VOID Render( UNIT *Uni, ANIM *Ani );
};
```

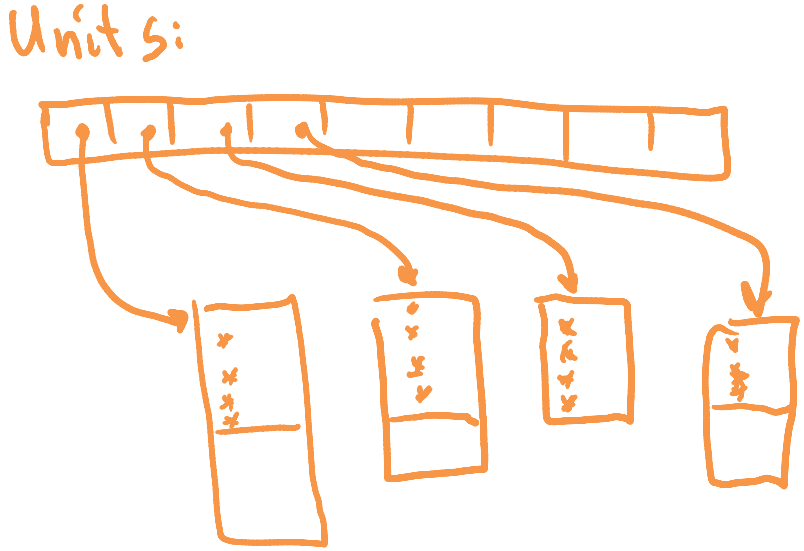
для коровы - свой тип:

```
typedef struct
{
  UNIT;
  OBJ3D Cow;
  VEC Pos;
} UNIT_COW;

UNIT_COW *Cow;
```

```
Cow->Init(&Cow, &Anim);
Cow->Pos = VecSet(5, 6, 7);

для деревьев:
typedef struct
{
  UNIT;
  VEC P[100];
  OBJ3D Tree;
} UNIT_TREES;
```

!!! Система анимации хранит массив УКАЗАТЕЛЕЙ на наши объекты, а так как в начале каждого - одинаковый тип UNIT - то она может пользовать Init, Close,. Response и Render

```
UNIT *Units[MAX];
```




```
Units[0] = CreateCow(...);
. . .
for (i = 0; i < NumOfUnits; i++)
  Units[i]->Render(Units[i], &Anim);
```

!!! для доступа к данным каждого экземпляра объекта - первым параметром у каждой функции является указатель на сам объект.