In C, we have

union as a structuring mechanism.

union int-or-float {
  int a,j;
  float b;
}
union int or float myVar;

myVar.a = 17.0;
printf("%.2f \n", myVar.a);
// 17 printed out
printf("%.2f \n", myVar.b);
// ...

-
myVar.b = 3.14 j

printf("\%of\n", myVar.b)

// 3.14 printed

printf("\%d\n", myVar.a);

// not 17

\n
Shrunk 3
int a, j
float b;
3 var 1, j

Unscre 3
int a, j
float b;
3 var 2, j

Var 1
a
b

Var 2
9, b

= vcr 5 = 17
= vcr 6 = 3.14
list

wasted
Shapes:

- Circle: $(x, y)$
- Rectangle: $(x_1, y_1)$
- $w, h$
- $x_2, y_2$
- $x_3, y_3$
- $x_4, y_4$

Types:

- Circle
- Rectangle
- Polygon
How do we create a list of shapes?

Payload

union { circle c, rectangle r, triangle t, polygon p, }
CurrentShape = currentShape \rightarrow \text{next};

// Draw currentShape;

// how do we know what
// kind of shape currentShape is?
// how do we draw the
// shape?
struct shape_data
{
    enum { circle, square, triangle, polygon }

    union
    {
        circle c;
        square s;
        triangle t;
        polygon p;
    }
}

shape-type = s_j
s.tag = circle_j
s.shape-data.c = --
```plaintext
else if (current_shape.tag == circle)
    tag = ring
    drawCircle(---);
else if (---.tag == square)
    drawSquare(---);
else if (---.tag == triangle)
    drawTriangle(---);
```
\[ r = \frac{P}{2\pi} \]

\[ C = 2\pi r = \pi d \]

Also if \( \theta \) is the central angle:

\[ \text{Area} = \frac{\theta}{360} \pi r^2 \]

\[ \text{Perimeter} = 2r + 2\pi r \]

Also, if \( \pi = \frac{C}{d} \)

\[ \ell = \sqrt{r^2 + c^2} \]
else is

\[ t_5 = r \times (\text{first}) \]

else if \( t_5 = c \times (\text{comp}) \)

else if \( t_5 = \sqrt{\text{comp}} \)
Polymorphism comes in handy... the computer rather than the programmer deals with causing the correct code to run.
In Java

```java
class Shape {
    public void draw() {
    }
}

class Circle extends Shape {
}

class Square extends Shape {
}
```
When Square extends Shape, that says Square obeys the same interface as Shape.
// to draw a shape in my list
CurrentShape.draw();

// what type of variable
// is currentShape?
Circle currentShape? NO!!!
Shape currentShape
We can cast up implicitly.

```java
public void someMethod() {
    Shape s;
    Circle c = new Circle();
    s = c;  // ok?
    Circle d;  // how many objects?
}
```
d = S \, \mathbf{j} \quad \text{// ok?}