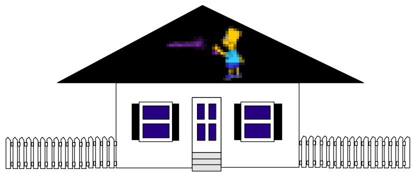
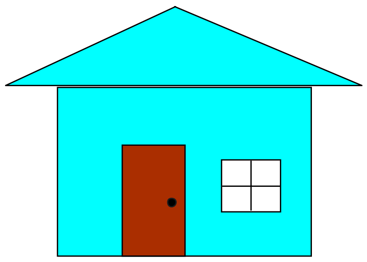
CSE / ENGR 142 Programming I

Functions and Design

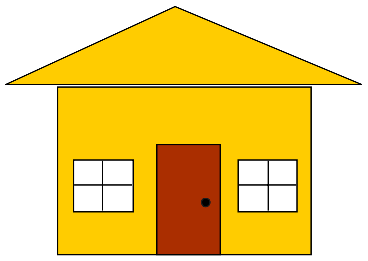
Drawing a House



Drawing a House



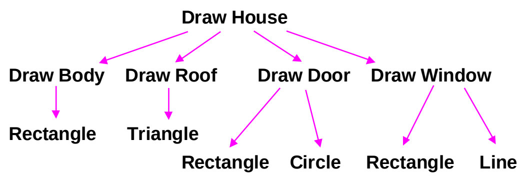
Drawing a (Similar) House



Draw House (Pseudo-code)

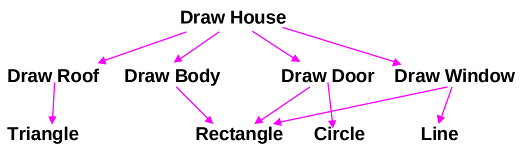
```
draw_house (color, ll_x, ll_y, num_windows)
    draw body as a colored rectangle
    draw roof as a colored triangle
    if num_windows is one
        draw door
        draw window
    if num_windows is two
        draw door
        draw window
        draw window
```

Functional Decomposition



This is a "calling tree" or "static call graph." Each function is shown, with an arrow down to each function called.

Functional Decomposition



Each function shown only once (preferred)

Analysis to Design to Programming

¶ Analyze the problem

¶ Then design a "big-picture" solution

  ¶ A functional decomposition shows how the pieces fit together

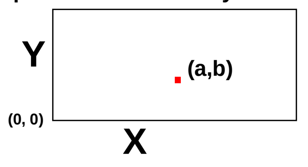
¶ Then design individual functions

  ¶ May depend on low-level ("primitive") functions available

¶ Final programming may be very detailed

Graphics Primitives

- Many systems offer a library of graphics primitives
  - –Typical functions: clearscreen, draw circle, rectangle, line, ellipse, etc.
  - –Typical parameters: location, color, fill, etc.
- Requires a coordinate system

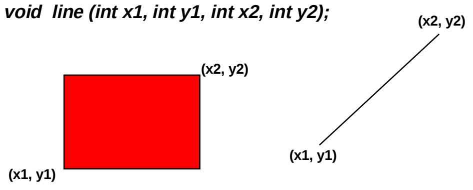


Typical ’rectangle’ and ’line’

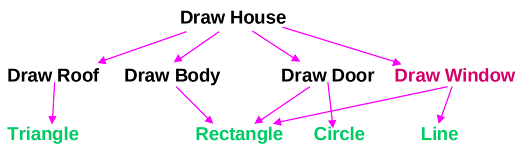
void

rectangle (int color, int x1, int y1, int x2, int y2);

void line (int x1, int y1, int x2, int y2);



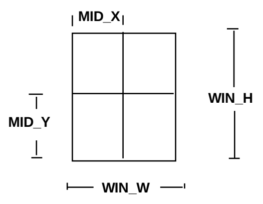
Big Picture Again



Fill in the pieces one at a time

Window Constants

Our analysis of how to describe a window

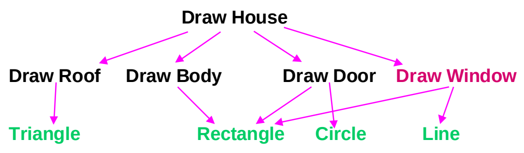


Map Analysis to C Code

- Identify and declare constants
- Choose parameters
- Utilize primitives
- Get the picky details right, too!

```
void draw_window(int x, int y)
    /* (x,y) is the lower left corner of the window */
{
    rectangle( WHITE,  x, y,  x + WIN_W, y + WIN_H);
    line( x+MID_X, y,    x + MID_X,   y + WIN_H);
    line( x,y + MID_Y,   x + WIN_W,  y + MID_Y);
}
```

Keep Filling in Pieces



Analyze and code remaining functions.
Does the order matter?

Draw House (gory details)

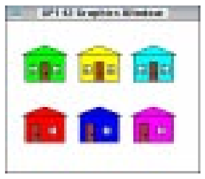
```
void draw_house (int color, int ll_x, int ll_y, int windows)
{
   int roof_ll_x, roof_ll_y ;

   /* Draw Body */
   draw_body (color, ll_x, ll_y) ;

   /* Draw Roof */
   roof_ll_x = ll_x - OVERHANG ;
   roof_ll_y = ll_y + BODY_HEIGHT ;
   draw_roof (color, roof_ll_x , roof_ll_y) ;

   /* Draw Door and Window(s) */
   if (windows == 1)
   {
      draw_door (ll_x + DOOR_OFFSET_1, ll_y) ;
      draw_window (ll_x + WINDOW_OFFSET_1,
                   ll_y + WINDOW_RAISE) ;
   }
   else if (windows == 2)
   {
      draw_door (ll_x + DOOR_OFFSET_2, ll_y) ;
      draw_window (ll_x + WINDOW_OFFSET_2A,
                   ll_y + WINDOW_RAISE) ;
      draw_window (ll_x + WINDOW_OFFSET_2B,
                   ll_y + WINDOW_RAISE) ;
   }
}
```

Next Step: A Neighborhood



We could write 6 different functions...

Smarter: call 1 function 6 times...

Summary of Functional Decomposition

- Look for common elements (similarities)
- Parameterize for special features (differences)
- Determine which functions will use others
  - Draw a graph to show their relationships

Review: Function Terminology

0! is 1
1! is 1
2! is 1 * 2
3! is 1 * 2 * 3
. . .

```
int factorial ( int n ) {
   int product, i ;
   product = 1 ;
   for ( i = n ;  i > 1 ;  i = i - 1 ) {
      product = product * i ;
   }
   return (product) ;
}
```

function name

[formal] parameter

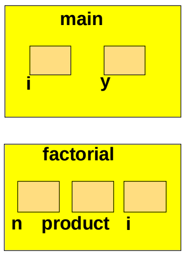
local variables

return type & value

Review: Local Variables

```
int
main(void)
{
    int i, y ;

    i = 3 ;
    y = factorial (i + 1) ;
    return (0) ;
}
```



Local Variables: Summary

- Formal parameters and variables declared in a function are local to it:
  - cannot be directly accessed by other functions
- Allocated (created) on function entry.
- De-allocated (destroyed) on function return.
- Formal parameters are initialized by copying value of actual parameter.
- *Reminder: no global variables in 142!*