

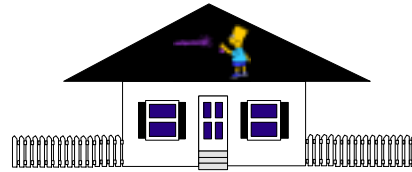
CSE / ENGR 142 Programming I

Functions and Design

© 1999 UW CSE

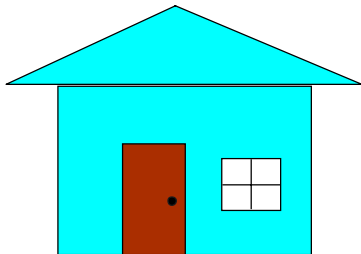
10/25/99 1-1

Drawing a House



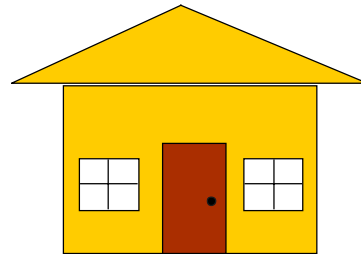
10/25/99 1-2

Drawing a House



10/25/99 1-3

Drawing a (Similar) House



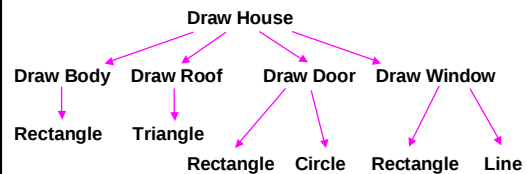
10/25/99 1-4

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
```

10/25/99 1-5

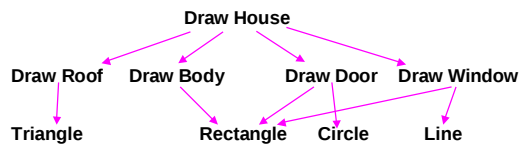
Functional Decomposition



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

10/25/99 1-6

Functional Decomposition



Each function shown only once (preferred)

10/25/99 1-7

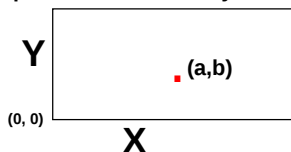
Analysis to Design to Programming

- 1 Analyze the problem
- 1 Then design a "big-picture" solution
 - 1 A functional decomposition shows how the pieces fit together
- 1 Then design individual functions
 - 1 May depend on low-level ("primitive") functions available
- 1 Final programming may be very detailed

10/25/99 1-8

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



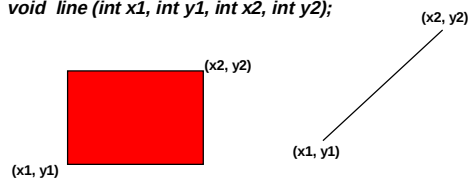
10/25/99 1-9

Typical 'rectangle' and 'line'

void

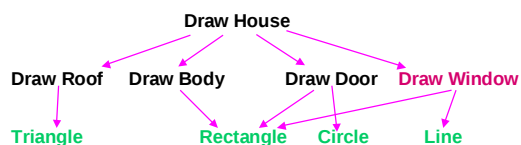
rectangle (int color, int x1, int y1, int x2, int y2);

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



10/25/99 1-10

Big Picture Again

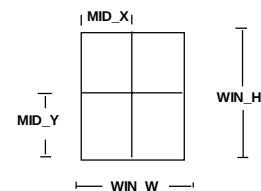


Fill in the pieces one at a time

10/25/99 1-11

Window Constants

Our analysis of how to describe a window



10/25/99 1-12

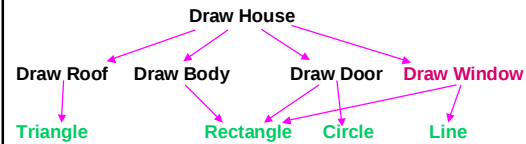
Map Analysis to C Code

- 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);
}
```

10/25/99 1-13

Keep Filling in Pieces



Analyze and code remaining functions.
Does the order matter?

10/25/99 1-14

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);
    }
}
```

10/25/99 1-15

Next Step: A Neighborhood



We could write 6 different functions...

Smarter: call 1 function 6 times...

10/25/99 1-16

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

10/25/99 1-17

Review: Function Terminology

```

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

return type & value
[formal] parameter
local variables
  
```

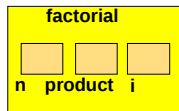
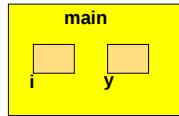
```

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

10/25/99 1-18

Review: Local Variables

```
int  
main(void)  
{  
    int i, y;  
  
    i = 3;  
    y = factorial (i + 1);  
    return (0);  
}
```



10/25/99 1-19

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!*

10/25/99 1-20