Lecture P2: Interacting with the OS

Some Elements of an Operating System

Files.

- Abstraction for storage (disks, DVD).
- File manipulation commands.

Processes.

- Abstraction for processor (CPU).
- Launching an application = initiating a process.

Interactions.

- Between user and machine.
- Among network of machines.
- Between files and processes.
- I/O redirection and pipes.

User Interface

Point and click.

- User launches applications by clicking.
  - `Compile` → `Project` → `hello.c`
- Restricted to pre-packaged menu options.

Command line.

- User types commands at terminal.
  `% gcc hello.c`
- Easily customizable.
  `% gcc126 hello.c`
- Extends to complex command sequences.

See "In the Beginning was the Command Line" by Neal Stephenson.

- http://www.spack.org/words/commandline.html

Files

File.

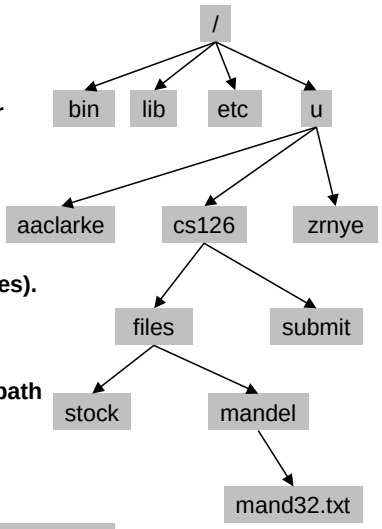
- Sequence of bits.
- A simple and powerful abstraction for permanent storage (disks).
- Extended for things beyond disks.

Directory.

- Sequence of files (and other directories).

Filename.

- Sequence of directory names on the path from "/" to the file.



Implements folder abstraction.

```
/u/cs126/files/mandel/mand32.txt
C:\cs126\files\mandel\mand32.txt
```

Processes

Process.

- An abstraction for the processor (CPU).
- Launching an application = initiating a process.

Multitasking.

- Illusion of multiple machines for your use.
  - abstraction provided by operating system
  - outgrowth of 1960s "time-sharing"
- Use it by opening one window for each application.
  - browser, editor, terminal window

Process Interconnection Abstractions

Standard input, standard output.

- Abstract files for command interfaces.

Redirection:

- Standard output to file.
  - `File → Save As`
- Standard input from file.
  - `File → Open`
  - drag-and-drop
- Standard input from file, standard output to file.

```
random > saveanswer        3
```

```
average < saveanswer
```

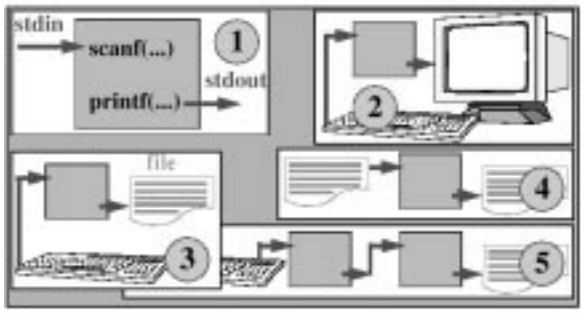
```
sort < myfile > myfilesorted        4
```

Piping:

- Connect standard output of one command to standard input of the next.
  - not easily expressed with point-and-click interface

```
random | average        5
```


I/O Redirection and Pipes

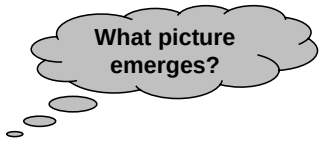


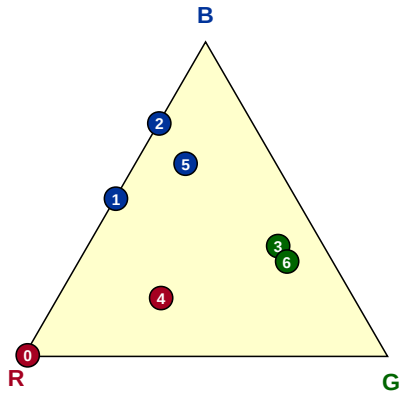
- 1: "Standard I/O", 2: default attachment, 3: redirect output
- 4: redirect both input and output, 5: pipes

Triangle Game

Game played on equilateral triangle, with vertices R, G, B.

- Start at R.
- Repeat the following:
  - pick a random vertex
  - move halfway between current point and vertex
  - draw a "dot" in color of vertex





Triangle Game: Boring Text Output

triangle.c

```
#include <stdio.h>
#define N 50000
#define SQRT3 1.732050808
int randomInteger(int n) { ... }

int main(void) {
   int i, r;
   double x = 0.0, y = 0.0, x0, y0;

   for (i = 0; i < N; i++) {
      r = randomInteger(3);
      if      (r == 0) { x0 =   0.0; y0 = 0.0;          }
      else if (r == 1) { x0 = 512.0; y0 = 0.0;          }
      else if (r == 2) { x0 = 256.0; y0 = 256 * SQRT3; }
      x = (x0 + x) / 2.0;
      y = (y0 + y) / 2.0;
      printf("%f %f\n", x, y);
   }
   return 0;
}
```


Turtle Graphics

ANSI C does not directly support graphical output.

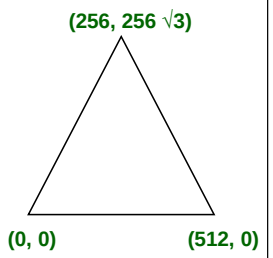
- Need help from OS.
- In this course we use "turtle graphics language."
  - render with PostScript, OpenGL, or Java

Turtle graphics.

- You command turtle to move, turn, and draw using relative coordinates.

Turtle Graphics Commands (relative)

```
D 512     // Walk 512 units, pen down
R 120     // Turn 120° counterclockwise
D 512
R 120
D 512
R 120
```



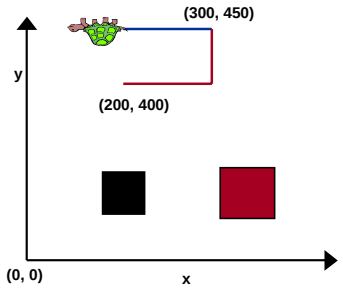
Flying Turtle Graphics

Flying turtle graphics.

- You also command turtle to fly to absolute coordinates and and drop colored spots below.

Turtle Graphics Commands

```
F 200 100  // Fly to (200, 100)
S 60       // Leave spot of size 60
F 400 100  // Fly to (400, 100)
C 1 0 0    // Change pen color to red
S 80       // Leave spot of size 80
F 200 400  // Fly to (200, 400)
D 100      // Walk 100 units, pen down
R 90       // Fly to (400, 100)
D 50
C 0 0 1
R 90
D 100
```



Triangle Game: Turtle Graphics Output

triangle.c

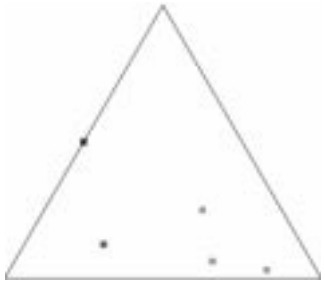
```
#include <stdio.h>
#define N 50000
#define SQRT3 1.732050808
int randomInteger(int n) { ... }

int main(void) {
   int i, r;
   double x = 0.0, y = 0.0, x0, y0;
   printf("F 0 0\n");
   printf("D 512 R 120 D 512 R 120 D 512 R 120\n");
   for (i = 0; i < N; i++) {
      r = randomInteger(3);
      if      (r == 0) { x0 =   0.0; y0 = 0.0;          }
      else if (r == 1) { x0 = 512.0; y0 = 0.0;          }
      else if (r == 2) { x0 = 256.0; y0 = 256 * SQRT3; }
      x = (x0 + x) / 2.0;
      y = (y0 + y) / 2.0;
      printf("F %f %f\n", x, y);
      printf("S 5\n");
   }
   return 0;
}
```

draw triangle

draw dot

Triangle Game: Turtle Graphics Output

Text output still boring!



Command Line

```
% echo 5 1.5 | triangle
F 0 0
D 512 R 120
D 512 R 120
D 512 R 120
F 128.000000 110.851252
S 1.500000
F 320.000000 55.425626
S 1.500000
F 160.000000 27.712813
S 1.500000
F 336.000000 13.856406
S 1.500000
F 424.000000 6.928203
S 1.500000
```

Write C program `turtle.c` to translate text into PostScript; render PostScript with `ghostscript`.

- Three pipes.
- See Assignment 2.

Command Line

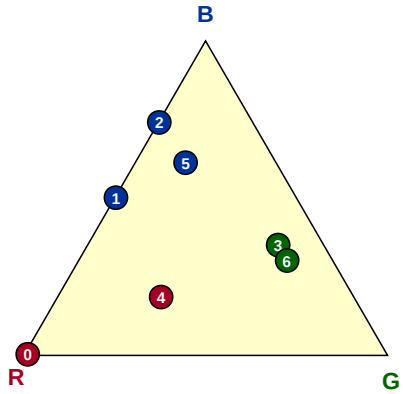
```
% echo 5 1.5 | triangle | turtle | gs -
```


Triangle Game

Game played on equilateral triangle, with vertices R, G, B.

- Start at R.
- Repeat the following:
  - pick a random vertex
  - move halfway between current point and vertex
  - draw a "dot" in color of vertex

What picture emerges?



Turtle Graphics in PostScript

turtle.c (includes and helper functions)

```
#include <stdio.h>
#include <math.h>
#define PI 3.1415926535897932385

// PostScript header
void header(void) {
    printf("%%!PS-Adobe-3.0 EPSF-3.0\n");
    printf("%%%%BoundingBox: 0 0 512 512\n");
}

// PostScript footer
void footer(void) {
    printf("showpage\n");
}
```

I'm a PostScript program.

Print me out.

Turtle Graphics in PostScript

turtle.c (main skeleton)

```
int main(void) {
    char command;           // input turtle command
    double d;               // distance
    double a;               // change in angle
    double r, g, b;         // new RGB colors

    double x = 0.0;         // x coordinate of turtle
    double y = 0.0;         // y coordinate of turtle
    double alpha = 0.0;     // turtle orientation

    header();

    // READ IN TURTLE GRAPHICS COMMANDS AND PROCESS

    footer();

    return 0;
}
```

Turtle Graphics in PostScript

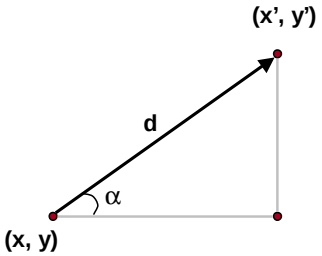
turtle.c (absolute coordinates)

```
// read in turtle graphics commands and process
while (scanf(" %c", &command) != EOF) {
   switch(command) {

      // change color to r, g, b
      case 'C': scanf("%lf %lf %lf", &r, &g, &b);
                printf("%f %f %f setrgbcolor\n", r, g, b);
                break;

      // fly to location (x, y)
      case 'F': scanf("%lf %lf", &x, &y);
                break;

      // draw spot of size d
      case 'S': scanf("%lf", &d);
                printf("%f %f %f %f rectfill\n",
                x - d/2, y - d/2, d, d);
                break;
```


Turtle Graphics: Relative Coordinates

Relative coordinates.

- Current location = (x, y).
- Current direction = α.
- Turtle moves d units.
- New location = (x', y').



Turtle Graphics in PostScript

turtle.c (relative coordinates)

```
      // rotate
      case 'R': scanf("%lf", &a);
                alpha += a;
                break;

      // move d units in current direction, pen down
      case 'D': scanf("%lf", &d);
                printf("%f %f moveto\n", x, y);
                x += d * cos((PI/180.0) * alpha);
                y += d * sin((PI/180.0) * alpha);
                printf("%f %f lineto\n", x, y);
                printf("stroke\n");
                break;

   }  // end switch
}     // end while
```