Announcements

- Exam #1
  - Was returned in section on Monday
  - Re-grade requests due by next Monday (yes first day of spring break)
- Reading
  - Chapter 16, 13.3 (Today)
    - Skip 16.4-16.6,16.9
  - Chapter 17 (Thursday)

Project #3 Hints

- Approach
  - Start with assembler
    - Skips labels until rest is working
  - Move onto execute
- Parsing
  - Use tokenize routine
- Labels
  - Think about how to handle
    - Labels defined before use
    - Labels defined after use

Additional Standard Library Functions

- To use, with gcc add –lm to link line
- Random Numbers (also in math.h)
  - void srand(unsigned int);
    - seed the random number generator
  - int rand(void);
    - return a pseudo-random number between 0 and RAND_MAX
- Floating Point (all use doubles)
  - prototypes defined in math.h
  - double sqrt(double value);
    - compute square root of value
    - Bad parameters produce domain errors
    - errno is set to EDOM
      - Errno is a global variable (externed in math.h)
      - Set to 0 to clear old error
      - Can be printed with perror
  - double exp(double value) and double exp2(double value)
    - computes e^{value} and 2^{value}

Trig Functions

- double sin(double angle);
  - computes sin of angle (in radians)
- Also have:
  - double cos(double angle);
  - double tan(double angle);
  - double asin(double value);
  - double acos(double value);
  - double atan(double value);

Other Handy Functions

- Fractions
  - double floor(double x);
    - next lowest whole integer
  - double ceil(double x);
    - next highest whole integer
  - double fabs(double x);
    - absolute value
  - double fmod(double x, double y);
    - restricts y to an integer

Function Pointers

- Pointers can also be of type function
  - void (*myVoidFunc)(int);
    - declare a variable myVoidFunc which points to a function which takes an integer as a parameter.
    - Like any pointer variable, declaring it does not create an instance of what it points to!
- Uses
  - Create Object Oriented Code
  - Callbacks from utility routines
- Often a good idea to typedef each function pointer
  - typedef void(*myVoidFuncPtr)(int);

Function Pointer Example

```
union myDataType {
    int a;
    float b;
} ;
typedef void (*myPrintFuncPtr)(union myDataType);

typedef struct {
    myPrintFuncPtr printIt;
    union myDataType data;
} myObject ;

void printInt(union myDataType data) {
    printf("data = %d\n", data.a);
}
```

Function Pointer Example Continued

```
int main(void) {
    int i;
    myObject *objects;

    objects = (myObject *) calloc(sizeof(myObject), 5);

    objects[0].data.a = 43;
    objects[0].printIt = printInt;
    objects[1].data.b = 3.1415;
    objects[1].printIt = printFloat;

    for  (i=0; i < 2; i++)  {
        objects[i].printIt(objects[i].data);
    }
    exit(0);
}
```

Callback Example

```
typedef int (*compareFunc)(item *a, item * b);


int searchTree(Node *root, Item *target, compareFunc cmp) {
        if (!root) return -1;
        ret = (cmp)(target, root->data);
          if (ret == 0) {
              /* found it */
              return 1;
          } else if (ret < 0) {
              return searchTree(root->left, target, cmp);
          } else {
             return searchTree(root->right, target, cmp);
          }
}
```

Date & Time Functions

- clock_t clock(void);
  - process time since start of program execution
  - to convert to time, use CLOCKS_PER_SEC
- time_t time(time_t *val);
  - fill val with the current time (in machine dependent format)
- char *ctime(time_t *val);
  - return a character representation of the passed time
  - Sun Jul 4 04:02:48 2005\n\0
- double difftime(time_t time1, time_t time2)
  - return number of seconds between time1 and time2
- struct tm *gmtime(time_t val)
  struct tm*localtime(time_t val)
  - convert to UTC or local time

Execution Environment

- Program Termination
  - abort(void);
    - terminate program with error (usually create a core file)
  - atexit(void (func)(void));
    - on termination call function func
  - exit(int status);
- Running Shell Commands
  - void system(char *command);
    - runs command (not all systems support it)
- Sorting
  - void qsort(void *base, size_t number, size_t elementSize, int (*compare)(void const *, void const *));
    - Sort the passed array, using passed compare function
    - strcmp will work for this!