Lecture 7
C Programming
Language
Summary of Lecture 7

• Libraries
• Recursion
• Unions
• time.h
Creating Libraries

• Assume you want to create a library that supports linked lists.

• Using the .c and .h files you wrote for defining a linked list and operations on a list, you compile them separately and then archive them into a library:

  % gcc -c -o link.o link.c
  % gcc -c -o list.o list.c

  % ar q mylistlib.a link.o list.o
  % ranlib mylistlib.a
Note: use “ar ruv” when library already exists.

• To use any of the functions in the .c files, include the appropriate header file and link mylist.a as follows:

  % gcc myprog.c mylistlib.a
Recursion

• Recursive Function - either directly or indirectly calls itself
• Serves as a tool to solve algorithms by reducing the original problem to a smaller problem (and reducing again…)
• Example:
  ```c
  int func1(int n) /* assumes n >= 0 */
  {
    if (n == 0)
      return 1;
    return (n * func1(n-1)); /* recursive call */
  }
  void func2(void) /* assumes user input */
  {
    int c;
    if ((c = getchar()) != ‘
      func2();
      putchar(c);
  }
  ```
Recursion - cont.

- The Towers of Hanoi:
  Given 3 poles, with disks in different sizes numbered 1..n according to size.
  Begin: all disks are stacked on pole A
    with disk 1 on top and disk n at bottom
  End: all disks are stacked on pole C in the same order

- move disk 1 from A to C
- move disk 2 from A to B
- move disk 1 from C to B
- move disk 3 from A to C
- move disk 1 from B to A
- move disk 2 from B to C
- move disk 1 from A to C

  ==> 7 moves for n = 3. (..15,31,63)
  - Move n-1 disks from A to B through C
  - Move disk n from A to C
  - Move n-1 disks from B to C through A
Recursion - cont.

- The Towers of Hanoi:

```c
void hanoi(int n, char *a, char *b, char *c) {
    if (n==1) {
        printf("Move disk 1 from %s to %s\n", a, c);
        return;
    }
    hanoi(n-1, a, c, b);
    printf("Move disk %d from %s to %s\n", n, a, c);
    hanoi(n-1, b, a, c);
}

main() {
    hanoi(3, "A", "B", "C");
    hanoi(6, "A", "B", "C");
}
```
Recursion - linked list

- Recursive function to create a lined list from an array of integers:

```c
Listitem * array_to_list (int *a, int size)
{
    Listitem * head;
    if (size == 0)
        return NULL;
    head = (Listitem *)malloc(sizeof(Listitem));
    head->data = a[0];
    if (size > 1)
        head->next = array_to_list(a+1, size-1);
    else
        head->next = NULL;
    return head;
}
```
Unions

- Unions are used as variables, when it’s convenient to have the same variable hold different types of data.
- In effect a union is a struct, in which all members have offset zero. The union is big enough to hold the largest member. It holds one member at a time.
- Example:
  ```c
  union int_or_float
  {
    int ival;
    float fval;
  }
  union int_or_float x;
  x.ival = 9;    // x as int
  x.fval = 4.321; // x as float
  // * overwrites int
  ```
Unions

• Example:

union int_or_float divide(int a, int b)
{
    union int_or_floar ans;

    if (a % b == 0)
        ans.ival = a/b;
    else
        ans.fval = a/(float)b;
    return ans;
}
...

divide(8,4);
divide(2,3);
Time.h

- This header file defines structures, macros and functions for manipulating date and time.
- Useful for timing your program

```c
typedef long clock_t;
typedef long time_t;
struct tm {
    int tm_sec;    /* seconds after the minute*/
    int tm_min;    /* minutes after the hour */
    int tm_hour;   /* hours after midnight */
    int tm_mday;   /* day of the month */
    int tm_mon;    /* months since January */
    int tm_year;   /* years since 1990 */
    int tm_wday;   /* days since Sunday */
    int tm_yday;   /* days since 1 January */
    int tm_isdst;  /*Daylight Savings Time flag */
};
```
Time.h

- `clock_t clock(void);`
  returns approximation of number of CPU clock ticks since beginning of execution. Use clock() / CLOCKS_PER_SECOND to convert to seconds.

- To measure time spent in program, call `clock()` at start of program, and its return value should be subtracted from subsequent calls.

- `time_t time(time_t *tptr);`
  returns current calendar time.

- `char *asctime(const struct tm *tp);`
  converts struct tm to a string, for printing

- `char *ctime(time_t *tptr);`
  converts time_t tptr to a string, for printing

- `double difftime(time_t t0, time_t t1);`
  returns t1 - t0

- Use two calls for `time` and then `difftime` to compute how long your program runs.