Announcements

- Program #1
  - Due in one week
- Email with
  - Schedule conflicts for midterm
  - DSS needs
- Reading
  - Chapter 7 & 8
    - skip 8.1.4
  - Chapter 9 & 10 (Thursday)

Switch Statement

```
switch (expression) {
   case constant-expression:
      statement;
      break;
   ....
   default:
       statement;
       break;
}
```

- Execution starts at the constant-expression equal to the expression
  - Without the break, will continue to next statement!!!
- Default constant expression matches anything else

Example of Switch Statement

```
switch (command) {
    case 'A':
            add_entry();
            break;
    case 'D':
            delete_entry();
            break;
    case 'P':
            print_entry();
            break;
    case 'E':
            edit_entry();
            break;
    default:
            printf("Error");
            break;
}
```

Pointer Variables

- Hold an address of something else
  - its location in memory
  - usually an address to a specified type of space
- type * varname;
  - int * a;
  - char * b;
  - int *a, b;
  - int *x, *y;
- Get an address by using a &
  - int a = 6;
  - printf("%d is at %p\n", a, &a);
- Dereference also using a star
  - int a, *b;
  - a = 7;
  - b = &a;
  - printf("%d and %d \n", a, *b);

Functions

- type name (parameters) block
  - type: return type for the function
  - name: name of the function
  - parameters: a comma separated list of types and names
  - block: the code to run
- return statement: return expression;
  - terminates a function at that point
  - expression is the value returned by the function

Function Example

```
int find(int data[10], int value)
{
        for (int i=0; i < 10; i++)  {
             if (data[i] == value)  {
                return i;
             }
        }
        return -1;
}
```

Function Arguments

- Comma separated list of values
- All parameters are passed by value
  - called function can modify values
  - arrays are passed by the address of their 0th element
    - modifying elements of array seen by calling function
- Can pass address of variable to change values

```
void foo(int *a) {
        *a = 3;
}
…
int x;
foo(&x);
…
```

Recursive Functions

- Functions that call themselves
  - can be indirect: a calls b calls a
- Example:

```
void binary_to_ascii(unsigned int value)
{
        unsigned int quotient;
        quotient = value / 10;
        if (quotient != 0) {
                binary_to_ascii(quotient);
        }
        printf("%c", value % 10 + '0');
}
```

Passing Parameters to Main

- int main(int argc, char *argv[])
  - Normal declaration of main
  - argc – number of command line arguments
    - Includes command itself
  - argv – array of character strings of command arguments

- Example
  - From command line: % myprog –file myfile
  - In the program:
    - argc == 3
    - argv[0] contains “myprog”
    - argv[1] contains “-file”
    - argv[2] contains “myfile”

Standard Type Definitions

- Several Types are defined in C's libraries
  - Defined in stddef.h
  - size_t - unsigned integer value (often int or long)
  - ptrdiff_t - signed value (used when subtracting pointers)
- Used by several standard routines
  - sizeof(<type>) returns a size_t for example

C-Strings

- Definition
  - An array of characters
  - Where the used portion is terminated by a null character
- <string.h>
  - Library that acts on C-strings
  - Most will crash if given something that does not fit the definition above
- Creating and Initializing a string

char name1[5] = {‘J’,’e’,’f’, ‘f’,’\0’};

char name2[14] = “Hollingsworth”;

- length of the string and the sizeof operator
  - sizeof operator tells the size of the variable or type
  - strlen uses the definition of C-string to find number of used characters

Strings

- Zero or more characters followed by null char '\0'
  - also called NUL
  - not counted as part of string
  - string.h defines prototypes for string routines
- Copying Strings
  - size_t strlen(char const *str);
    - returns count of characters in str
    - up to but not including the null character
  - char *strncpy(char *dst, char const *src, size_t len);
    - copy src to dst
    - copy until '\0' in src or at most len characters
    - pad extra characters will '\0'
    - Safety tip: dst[len-1] = '\0'; to force termination of new string
  - char *strncat(char *dst, char const *src, size_t len);
    - append src onto the end of dst
    - always appends NUL to end of dst string

String Functions

- Comparison
  - int strncmp(char const *s1, char const *s2, size_t len);
    - returns 0 if string equal up to len
    - returns a negative value if s1 is less than s2
    - returns a positive value if s1 is greater than s2
- Searching
  - char *strchr(char const *str, int ch);
  - char *strrchr(char const *str, int ch);
    - finds the first occurance of ch in str
    - strrchr finds the last occurance
    - returns NULL if not found
  - char *strstr(char const *s1, char const *s2);
    - find the first occurrence of s2 in s1

String Functions Examples

```
char string[] = "this is a test string";
char *ans;
int length;

length = strlen(string);          /* returns 21 */
ans = strchr(string, 'h');        /* returns string + 1 */
ans = strrchr(string, 't');       /* returns string + 16 */
ans = strstr(string, "test");     /* returns string + 10 */
```