Lecture 6. Strings

- **A *string* is an array of characters; quotes enclose *string constants***

```
/* Everyone’s first
    C program. */
#include <stdio.h>

int main(void) {
    char hello[13] = { 'H', 'e', 'l', 'l', 'o', ' ',
        'W', 'o', 'r', 'l', 'd', '!', '\0' };

    printf("%s\n", hello);
    return 0;
}
```

A strings is terminated with a *null character* — the character with value 0

The *conversion specifier* %s causes the value of the corresponding string argument to be printed instead; i.e., its characters up to the null character

- **Strings can be initialized with individual characters as above, or by**

```
char hello[] = "Hello World!";
char *hello = "Hello World!";
```

let the compiler count the characters

`char *` declares a *character pointer*, which — for now — is the same as a string

- **String variables can be used anywhere constant strings can be used**
- **Elements of string variables — the characters — can be changed by assignments**

Printing Repeated Words

```
% lcc double.c
% echo Now is the the time | a.out
the
%
```

```
/* Print repeated words. */
#include <stdio.h>
#include <ctype.h>
#include <string.h>

int main(void) {
    char prev[100], word[100];

    prev[0] = '\0';
    while (scanf("%s", word) != EOF) {
        if (isalpha(word[0]) && strcmp(prev, word) == 0)
            printf("%s\n", word);
        strcpy(prev, word);
    }
    return 0;
}
```

Dissecting double.c

```
#include <ctype.h>
#include <string.h>
```

Includes the declarations for the character handling functions (`ctype.h`) and the string handling functions (`string.h`)

```
char prev[100], word[100];
```

```
prev[0] = '\0';
```

Declares two strings, `prev` and `word`, each capable of holding up to 100 characters, and initializes `prev` to the *empty string*

```
while (scanf("%s", word) != EOF) {
    …
}
```

Loops reading the next string of nonblank characters into `word`

```
    if (isalpha(word[0]) && strcmp(prev, word) == 0)
        printf("%s\n", word);
    strcpy(prev, word);
```

Prints `word` if it begins with a letter (`isalpha`) and holds the same word as `prev`; `strcmp` compares strings; then copies the string in `word` into `prev` (`strcpy`)

`strcmp(x, y)` returns a value <0, =0, >0 if `x` < `y`, `x` == `y`, `x` > `y` (lexicographic order)

Implementing String Handling Functions

- **`strcpy(dst, src)` copies `src` to `dst`, character-by-character up to the `’\0’`**

```
void strcpy(char dst[], char src[]) {
        int i;

        for (i = 0; src[i] != ’\0’; i++)
                dst[i] = src[i];
        dst[i] = ’\0’;
}
```

- **`strcmp(str1, str2)` compares `str1` and `str2`, character-by-character**

```
int strcmp(char str1[], char str2[]) {
        int i;

        for (i = 0; str1[i] == str2[i] && str1[i] != ’\0’; i++)
                ;
        if (str1[i] < str2[i])
                return -1;
        else if (str1[i] > str2[i])
                return +1;
        else
                return 0;
}
```

- **Other string handling functions**

`strlen(str)`	**returns the number of nonnull characters in `str`**
`strcat(dst, src)`	***appends* `src` to the *end* of `dst`**

Arrays of Strings

```
/* Shuffle a deck of cards. */
#include <stdio.h>
#include <stdlib.h>

char *suits[] = {
    "Hearts", "Diamonds", "Clubs", "Spades"
};

char *faces[] = {
    "Ace", "2", "3", "4", "5", "6", "7", "8",
    "9", "10", "Jack", "Queen", "King"
};

int main(void) {
    int i, deck[52];

    deck[0] = 0;
    deck[1] = 1;
    for (i = 2; i < 52; i++) {
        int k = rand()%i;
        deck[i] = deck[k];
        deck[k] = i;
    }
    for (i = 0; i < 52; i++)
        printf("%s of %s\n", faces[deck[i]%13], suits[deck[i]/13]);
    return 0;
}
```

```
% lcc shuffle.c
% a.out
3 of Diamonds
2 of Spades
Jack of Hearts
7 of Spades
9 of Clubs
Ace of Clubs
6 of Clubs
…
6 of Hearts
Ace of Diamonds
4 of Spades
10 of Spades
5 of Clubs
…
King of Spades
8 of Clubs
Queen of Clubs
8 of Spades
%
```

Dissecting shuffle.c

- **Integer k (0..51) represents the card with face value `k%13` (0..12) and suit `k/13` (0..3)**

```
char *suits[] = {
    "Hearts", "Diamonds", "Clubs", "Spades"
};

char *faces[] = {
    "Ace", "2", "3", "4", "5", "6", "7", "8",
    "9", "10", "Jack", "Queen", "King"
};
```

Define and initialize global arrays of strings that map integers to suits and faces

```
deck[0] = 0;
deck[1] = 1;
for (i = 2; i < 52; i++) {
    int k = rand()%i;
    deck[i] = deck[k];
    deck[k] = i;
}
```

Initializes `deck[0..51]` to a random *permutation* of the integers 0..51

```
for (i = 0; i < 52; i++)
    printf("%s of %s\n", faces[deck[i]%13], suits[deck[i]/13]);
```

Prints the permuted deck in a readable form by mapping `deck[i]%13` (0..12) to a face and `deck[i]/13` (0..3) to a suit

Command-Line Arguments

- **By convention, `main` is called with two arguments**

```
int main(int argc, char *argv[])
```

`argc`	**(‘*argument count*’) is the number of command-line arguments, including the program name**
`argv`	**(‘*argument vector*’) is an array of strings, one for each argument**

```
% echo Hello World
Hello World
%
```

- **Implementing echo**

```
/* Echo my arguments. */
#include <stdio.h>

int main(int argc, char *argv[]) {
    int i;

    if (argc > 1)
        printf("%s", argv[1]);
    for (i = 2; i < argc; i++)
        printf(" %s", argv[i]);
    printf("\n");
    return 0;
}
```

```
% lcc echo.c
% a.out Hello World
Hello World
%
```

Inside `main`:

```
argc = 3
argv[0] = "a.out"
argv[1] = "Hello"
argv[2] = "World"
```

Testing random()

- **Check `argc` for optional command-line arguments**

```
/*
Use random() to generate N (default 100)
random numbers, perhaps with different seeds.
*/
#include <stdio.h>
#include "random.h"

int main(int argc, char *argv[]) {
    int n = 100;

    if (argc > 1)
        sscanf(argv[1], "%d", &n);
    if (argc > 2)
        sscanf(argv[2], "%d", &seed);
    while (n-- > 0)
        printf("%d\n", random());
    return 0;
}
```

sscanf is like scanf, but reads from a string instead of from the input

```
% lcc testrandom.c random.c
% a.out | fmt
520932930 28925691 822784415 890459872 ...
```
... 100 random numbers

```
% a.out 1000 | fmt
520932930 28925691 822784415 890459872 ...
```
... 1000 random numbers

```
% a.out 4 126217318 | fmt
2088403071 1317687729 1526293439 721665858
```