3/18/14

Library Strings Functions
Based on slides from K. N. King and Dianna Xu
Bryn Mawr College
CS246 Programming Paradigm

Using the C String Library
• Strings are treated as arrays in C.
• Strings can’t be copied or compared using operators.
  char str1[10], str2[10];
  str1 = "abc"; /* WRONG */
  str2 = str1; /* WRONG */
  if(str2 == str1) /* WRONG */
Since str1 and str2 have different addresses, the expression str1 == str2 must have the value 0.
Using an array name as the left operand of = is illegal.
• Initializing a character array using = is legal, though:
  char str1[10] = "abc";
In this context, = is not the assignment operator.

Library String Functions
• #include <string.h>
• Find the Length of a string
  size_t strlen(const char *str)
• Copy a string (including the '\0')
  char *strcpy(char *t, const char *s)
• Concatenate two strings
  char *strcat(char *t, const char *s)
• Compare two strings
  int strcmp(const char *s1, const char *s2)
  Return 0 if identical, ASCII difference between the first mismatch otherwise.
  "abc" vs "abc": +1;  "abc" vs "abc": -1

Example
int main() {
  char s[] = "ann"; char s2[] = "abby";
  char s3[strlen(s)+strlen(s2)+1];
  printf("%d\n", strlen(s));
  printf("%d\n", strlen(s2));
  strcpy(s3, s);
  strcat(s3, s2);
  printf("%s\n", &s3[2]);
  printf("%d\n", strcmp(s, s2));
  return 0;
}

Length Function strlen
size_t strlen(const char *s) {
  size_t n;
  for (; n != 0; *s != '\0'; s++)
    n++;
  return n;
}
Length Function `strlen`

```c
size_t strlen(const char *s) {
    size_t n = 0;
    for (; *s; s++)
        n++;
    return n;
}
```

Length Function `strlen`

```c
size_t strlen(const char *s) {
    size_t n = 0;
    for (; *s++; )
        n++;
    return n;
}
```

Length Function `strlen`

```c
size_t strlen(const char *s) {
    size_t n = 0;
    while (*s++)
        n++;
    return n;
}
```

Length Function `strlen`

```c
const char *p = s;
while (*s)
    s++;
return s - p;
```

Searching for the End of a String

```c
while (*s)     while (*s++)
    s++;           ;
```

- The first version leaves `s` pointing to the null character.
- The second version is more concise, but leaves `s` pointing just past the null character.

Copy Function `strcpy`

```c
char* strcpy(char to[], char from[]) {
    int i;
    for (i = 0; from[i] != '\0'; i++)
        to[i] = from[i];
    to[i] = '\0'; return to;
}
```

```c
char* strcpy(char *to, char *from) {
    char *tmp = to;
    while ((*to = *from) != '\0'){
        to++; from++;
    }
    return tmp;
}
```
strcpy, pointer version

```c
char* strcpy(char *to, char *from) {
    char *tmp = to;
    while ((*to++ = *from++) != '\0');
    return tmp;
}
```

char* strcpy(char *to, char *from) {
    char *tmp = to;
    while (*to++ = *from++);
    return tmp;
}

strcat

```c
char* strcat(char *s1, const char *s2) {
    char *p = s1;
    while (*p != '\0')
        p++;
    while (*s2 != '\0') {
        *p = *s2;
        p++;
        s2++;
    }
    *p = '\0';
    return s1;
}
```

Condensed version strcat

```c
char *strcat(char *s1, const char *s2) {
    char *p = s1;
    while (*p)
        p++;
    while (*p++ = *s2++)
        ;
    return s1;
}
```

• What causes the loop to terminate?

Comparison strcmp

```c
int strcmp(char *s, char *t) {
    for(;*s == *t; s++,t++) {
        if (*s == '\0')
            return 0;
    }
    return *s-*t;
}
```

• Returns 0 if s == t
• If not, returns the difference btw the first chars that differ

Cannot compare strings using ==
Other Library String Functions

- **`n` functions**
  - `char *strncpy(char *t, const char *s, size_t n)`
  - `char *strncat(char *t, const char *s, size_t n)`
  - `int strncmp(const char *s1, const char *s2, size_t n)`
  - Same as the none-`n` functions, only works on `n` chars

- A safer way to use `strncpy`:
  - `strncpy(str1, str2, sizeof(str1) - 1);`
  - `str1[sizeof(str1)-1] = '\0';`

- Search for a character in a string
  - `char* strchr(char *s, char c)`

- Search for a (sub)string in a string
  - `char* strstr(char *s, char *substr)`

```
string.h Functions Example

int main() {
    char s[] = "abcdejkljikl", s2 = 'b';
    char *s3 = "jkl", *s4, *s5;
    s4 = strchr(s, s2);
    s5 = strstr(s, s3);
    printf("%s\n", s4);
    printf("%c\n", s5[5]);
    printf("%d\n", s5-s4);
    printf("%s\n", &s[2]);
}
```

- Using library functions

```
Using library functions

int main() {
    char s1[] = "The name is Bond";
    char s2[] = "Bond, James Bond";
    char s3[100];
    strcpy(s3, s1, 12);
    s3[12] = '\0';
    strncat(s3, &s2[6], 5);
    printf("%s\n", s3);
}
```

- Remember to always null-terminate a string
- `string.h` functions may have undefined behaviors otherwise

Arrays of Strings

- Array of strings ==> two-dimensional character array, with one string per row:
- The number of rows in the array can be omitted, but we must specify the number of columns.

```
Arrays of Strings

int main() {
    char s1[] = "The name is Bond";
    char s2[] = "Bond, James Bond";
    char s3[100];
    strcpy(s3, s1, 12);
    s3[12] = '\0';
    strncat(s3, &s2[6], 5);
    printf("%s\n", s3);
}
```

- Unfortunately, the `planets` array contains a fair bit of wasted space (extra null characters):
Arrays of Strings

- Most collections of strings will have a mixture of long strings and short strings.
- In a ragged array, rows can have different lengths.
- We can simulate a ragged array in C by creating an array whose elements are pointers to strings:
  ```c
  ```

Arrays of Strings

- This small change has a dramatic effect on how `planets` is stored:

```
for (i = 0; i < 9; i++)
  if (planets[i][0] == 'M')
    printf("%s begins with M
", planets[i]);
```

Command-Line Arguments

- Available through two parameters to `main`:
  - `main(int argc, char *argv[])`
  - `argc` – argument count, i.e. number of args
  - `argv` – an array of pointers to the arguments

```
for (p = &argv[1]; *p != NULL; p++)
  printf("%s\n", *p);
```