(9-3) Strings II
H&K Chapter 8

Instructor – Beiyu Lin
CptS 121 (May 30th, 2019)
Washington State University

String Manipulation in C (1)

- Standard operators applied to most numerical (including character) types cannot be applied to strings in C
  - The assignment operator (=) can't be applied except during declaration
  - The + operator doesn't have any true meaning (in some languages it means append)
  - The relational operators (==, <, >) don't perform string comparisons
  - Others?

C. Hundhausen, A. O'Fallon

String Manipulation in C (2)

- The string-handling library <string.h> provides many powerful functions which may be used in place of standard operators
  - strcpy ( ) replaces the assignment operator
  - strcat ( ) replaces the + or append operator
  - strcmp ( ) replaces relational operators
  - Several others…

C. Hundhausen, A. O'Fallon

String Library Functions (1)

- Provided the following code fragment:

```
char string1[20] = "foo", string2[20];
string2 = string1; /* Will cause compiler error! */
if (string1 == string2) /* illegal! */
  printf("The strings are equal!");
```

- In order to perform assignment and comparison, we need to call upon C library functions…

C. Hundhausen, A. O'Fallon

String Library Functions (2)

- To use string library functions, we need to include the string library:

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

- String assignment

```
strcpy(char *dest, const char *source);
```

  - Creates a fresh copy of `source`, assigning it to `dest` (an output parameter)
  - Example:

```
char string1[20] = "CptS 121", string2[20];
strcpy(string2,string1);
/* string2 now also contains "CptS 121\0" */
```


C. Hundhausen, A. O'Fallon

String Library Functions (3)

- String assignment: An alternative

```
strncpy(char * dest, const char *source,
        unsigned int size_t);
```

  - Copies `size_t` characters of source to `dest`. If `source` has fewer than `size_t` characters allocated to it, then `'\0'` is used to fill the remaining characters
  - Example:

```
char string1[20] = "CptS", string2[20];
strncpy(string2,string1,6);
/* string2 now contains "CptS\0\0" */
```


C. Hundhausen, A. O'Fallon

String Library Functions (4)

- String assignment: Only copying as many characters as will fit in the destination string
  - Notice that, if we're not paying attention, it's easy to copy more characters to the destination string than the space available:

```
char string1[20] = "CptS", string2[4];
strncpy(string2,string1,6);
/* string2 now also contains "CptS\0\0", but the
   null terminators are beyond string2's
   allocated memory, so the string isn't properly
   terminated! */
```

  - We can guard against this situation by making sure that we copy one fewer characters than the length of the `dest` string:

```
strncpy(dest, source, dest_len - 1);
dest[dest_len - 1] = '\0';
```


C. Hundhausen, A. O'Fallon

String Library Functions (5)

- String assignment: substrings
  - We often want to extract a series of characters from a longer string
    - Example: Extract the elements of a date string like "26-Oct-2020"
    - Example: Piece together a person's initials from three strings containing the first, middle, and last name
  - We can use `strncpy` to perform this task
  - Example: Extract: 121 from the string `"CptS 121 is awesome!"`

```
char string1 [25] = "CptS 121 is awesome!",
     string2[4];
strncpy(string2, &string1[5],3);

/* string2 now contains "121\0" */
```


C. Hundhausen, A. O'Fallon

String Library Functions (6)

- String assignment: substrings (cont.)
  - We can also use the `strtok` function from the C string library to extract substrings. Suppose, for example, that we want to extract the department code ("CptS"), course number (“121”), semester (“Fall”), and year (“2018”) elements from "CptS 121, Fall, 2018"

```
char course[25] = "CptS 121, Fall, 2018";
char course_copy[25];
char *dept_code, *course_num, *semester, *year;
strcpy(course_copy,course); /*strtok alters source string */
dept_code = strtok(course_copy,", ");
course_num = strtok(NULL,", ");
semester = strtok(NULL,", ");
year = strtok(NULL,", ");
printf("dept. code: %s\n",dept_code);
printf("course num: %s\n",course_num);
printf("semester: %s\n",semester);
printf("year: %s\n",year);
```

Output:

```
dept. code: CptS
course num: 121
semester: Fall
year: 2018
```


C. Hundhausen, A. O’Fallon

String Library Functions (7)

- String assignment: concatenation
  - Now suppose that we want to do just the opposite: Given the substrings "CptS", "121", “Fall", and "2018", we want to piece together the bigger string "`CptS 121, Fall, 2018`"
    We can use the C string library `strcat` and `strncat` functions to do this task:

```
char course[25] = ""; /* Empty string—first char is '\0' */
char dept_code[5] = "CptS";
char course_num[7] = "121 is";
char semester[7] = “Fall";
char year[5] = "2018";
strcat(course,dept_code);
strcat(course," ");
strncat(course,course_num,3); /* Take only 1st 3 char */
strcat(course,", ");
strcat(course,semester);
strcat(course,", ");
strcat(course,year);
printf("%s\n",course);
```

Output:

```
CptS 121, Fall, 2018
```


C. Hundhausen, A. O’Fallon

String Library Functions (8)

- String comparison

```
int strcmp(const char *s1, const char *s2);
```

  - Returns
    - a *negative* value if s1 < s2
    - 0 if s1 == s2
    - a *positive* value if s1 > s2
  - Example:

```
char string1[20] = "CptS 121",
     string2[20] = "CptS 122";
int result = strcmp(string1,string2);
/* result now contains a negative value */
```

  - Note: To determine whether one string is greater than, less than, or equal to another, `strcmp` compares the strings, one character at a time
    - All comparisons are based on ASCII character codes

C. Hundhausen, A. O'Fallon

You Try It (1)

Re-write the Selection Sort Algorithm on the next slide so that it operates on an array of strings. Test your code on sample input data sets of strings.

C. Hundhausen, A. O'Fallon

You Try It (2)

- Code for Selection Sort

```
void selection_sort(int values[], int num_values)
{
      int i, index_of_smallest, temp;
      for (i = 0; i < num_values - 1; ++i)
      {
              /* Find the index of the smallest element in unsorted list... */
              index_of_smallest = find_smallest(values,i,num_values-1);
              /* Swap the smallest value in the subarray i+1 .. num_values - 1
                 with the value i, thereby putting into place the ith element. */
              temp = values[i];
              values[i] = values[index_of_smallest];
              values[index_of_smallest] = temp;
      }
}

int find_smallest(int values[], int low, int high)
{
      int smallest_index, i;
      smallest_index = low;
      for (i = low + 1; i <= high; i++)
              if (values[i] < values[smallest_index])
                              smallest_index = i;
      return smallest_index;
}
```


C. Hundhausen, A. O'Fallon

Next Lecture…

- Structures

References

- J.R. Hanly & E.B. Koffman, *Problem Solving and Program Design in C (8th Ed.)*, Addison-Wesley, 2016
- P.J. Deitel & H.M. Deitel, *C How to Program (7th Ed.)*, Pearson Education , Inc., 2013.

Collaborators

- Chris Hundhausen
- Andrew O' Fallon