Programming Fundamentals

Module G - Strings

Le The Anh
anhlt161@fe.edu.vn

FPT Education
FPT UNIVERSITY

Objectives

1. Strings
   - Null Byte Terminators
   - Declaration
   - Iteration
   - Output
2. String Input
   - scanf
   - gets
3. String Functions and Arrays of Strings
   - String Functions
   - Arrays of Strings
   - In-Class Practice

Strings and Null Bytes

- A string is an array of characters, terminated by a null-byte.
- The null byte is the character with the value 0. All of the bits in the null byte are 0's.
- The null byte occupies the first position in the ASCII collating sequence.
- The null byte is used to locate the last meaningful element in a string. We store the null byte immediately after the last meaningful element.

Byte	1	2	3	4	5	6	7	8	9	10	11	12	13
Character	H	i		e	v	e	r	y	o	n	e	!	\0

Declaration

- A string is declared in the same way that we declare a character array, except that we allocate space for the terminator null byte. For example, if we want a string to hold up to 30 non-null characters, we declare

```
char name[31]; /* room for 30 characters */
```

- To initialize a string, we can use one of the two ways below:

```
char name[31] = "Garry Kasparov";
```

```
char name[31] = {'G', 'a', 'r', 'r', 'y', ' ',
                 'K', 'a', 's', 'p', 'a', 'r', 'o', 'v', '\0'};
```

Iteration

- In an iteration, we can check for the presence of the null byte directly rather than the number of characters in the string.

```
int i;
char name[31] = "Garry Kasparov";
for (i = 0; name[i] != '\0'; i++)
    printf("%c", name[i]);
```

- A non-terminated character array requires a separate integer to hold the number of characters in the array.

```
void print(char str[], int n)
{
    int i;
    for (i=0; i < n; i++)
        printf("%c", str[i]);
}

int main()
{
    char name[31] = {'G', 'a', 'r', 'r', 'y', ' ',
                     'K', 'a', 's', 'p', 'a', 'r', 'o', 'v'};
    print(name, 14);
    return 0;
}
```

Output

- `printf` uses the conversion specifier %s to process strings.

```
char name[31] = "Garry Kasparov";
printf("%s", name);
```

Summary

- Strings
  - Null Byte Terminator
  - Declaration
  - Output

Q&A

In-Class Practice

- Add code to the following program to display the string "`This is a string`". Allocate memory for the string and initialize that memory to hold the phrase. Pass the string to the display function and in that function send the string to the standard output stream so that the string appears left-justified in a field of 20 characters.

```
#include <stdio.h>

void display(...)
{
    ...
}

int main ()
{
    display(...);

    return 0;
}
```

scanf

- `scanf` accepts two conversion specifiers: %s, %[^\n].
- %s conversion specifier
  - reads all characters until the first whitespace character,
  - stores the characters read in memory locations starting with the address passed to scanf,
  - stores the null byte in the memory byte following the last character accepted and
  - leaves the delimiting whitespace plus any subsequent characters in the input buffer.
- %[^\n] conversion specifier
  - reads all characters until the newline (\n),
  - stores the characters read in memory locations starting with the address passed to scanf,
  - stores the null byte in the byte following that where scanf stored the last character and
  - leaves the delimiting character (here, \n) in the input buffer.

scanf - Examples

```c
char s1[31], s2[31], s3[31], s4[31];
char s5[31], s6[31], s7[31], s8[31];

scanf("%s", s1); // Garry Kasparov -> Garry
fflush(stdin);
scanf("%3s", s2); // Garry Kasparov -> Gar
fflush(stdin);
scanf("%[^\n]", s3); // Garry Kasparov -> Garry Kasparov
fflush(stdin);
scanf("%[0-9]", s4); // 012a3 -> 012
fflush(stdin);
scanf("%[^0-9]", s5); // abc1d -> abc
fflush(stdin);
scanf("%[0-3ab]", s6); // 01acb -> 01a
fflush(stdin);
scanf("%3[0-3ab]", s7); // 012ab -> 012
fflush(stdin);
scanf("%[0-3][^a-b]", s8); // 01acb -> 01

printf("\n%s\n%s\n%s\n%s", s1, s2, s3, s4);
printf("\n%s\n%s\n%s\n%s", s5, s6, s7, s8);
```

gets

- gets is a standard library function that
  - accepts an empty string
  - uses the '\n' as the delimiter
  - throws away the delimiter after accepting the string
  - appends the null byte to the end of the set stored
- gets is dangerous. It can fill beyond the memory that allocated for the string. Nevertheless, gets is an excellent tool for quick and dirty input.

```
char s[20];
gets(s);
```

String input using getchar()

```c
#define MAX 30
#include <stdio.h>

void getstr(char s[], int max);

int main () {
    char string[MAX+1];
    do {
        printf("Enter a string (empty to quit) : ");
        getstr(string, MAX);
        printf("You entered ==>%s<==\n", string);
    } while (string[0] != '\0');

    return 0;
}

/* getstr accepts a newline terminated string s of up
*  to max characters, appends a null byte and throws
*  away the terminating character
*/
void getstr(char s[], int max) {
    int i, c;

    i = 0;
    while((c = getchar()) != '\n' && c != EOF)
        if (i < max)
            s[i++] = (char) c;
    s[i] = '\0';
}
```

In-Class Practice

- Write a function to take as input a string and convert it to uppercase (Try to use some function in the ctype.h library)

String Functions

- String is an array of char. Array names alone refer to addresses rather than the values stored in the array elements. Accordingly, we cannot apply the operators: $=, +, <, >$
- The string library <string.h> contains the functions for string manipulation:
  - strlen - for finding the number of characters in a string
  - strcpy - for copying one string to another
  - strcmp - for comparing two strings
  - strcat - for concatenating one string to another

strlen

- `strlen` returns the number of characters in the string excluding the null byte terminator. In other words, `strlen` returns the index of the null byte.
- Prototype:
  ```
  unsigned int strlen(char []);
  ```
- Example: print the string in reverse order

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

int main ( ) {
    char str[31];
    int i, len;

    printf("Enter a string   : ");
    scanf("%30[^\n]%*c", str);
    printf("In reverse order : ");
    len = strlen(str);
    for (i = len - 1; i >= 0; i--)
        putchar(str[i]);
    putchar('\n');

    return 0;
}
```

strcpy

- `strcpy` copies `source` into `destination` and returns the address of `destination`. It is our responsibility to ensure that there is sufficient space in `destination` to hold all of the characters in `source`.
- Prototype:

```
char* strcpy(char destination[], char source[]);
```

- Example

```
char str[31], copy[21] = "?";
int i, len;

printf("Enter a string   : ");
// The * in a scanf() format means 'read the data but do not assign
// it to a variable in the argument list'
scanf("%30[^\n]%*c", str);
len = strlen(str);
if (len < 21) {
    strcpy(copy, str);
    printf("Copy contains : %s\n", copy);
} else {
    printf("Not enough room for a full copy\n");
    printf("Copy contains : %s\n", copy);
}
```

strcmp

- `strcmp` is used to compare two strings. It returns
  - 0 if the strings are identical,
  - a negative number if the first non-matching character in the first string is lower than the character with the same index in the second string,
  - a positive number for otherwise.
- Prototype:

```
int strcmp(char source1[], char source2[]);
```

- Example

```
char s1[] = "acd", s2[] = "acd", s3[] = "aec", s4[] = "abc";
printf("%d ", strcmp(s1, s2));
printf("%d ", strcmp(s1, s3));
printf("%d", strcmp(s1, s4));

// Output: 0 -2 1
```

strcat

- `strcat` appends `source` to `destination` and returns the address of `destination`. It is our responsibility to ensure that there is sufficient space in `destination` to include both the original string and the appended source.
- Prototype:

```
char* strcat(char destination[], char source[]);
```

- Example

```
char givenName[31], surname[31], fullName[62];

printf("Enter given name  : ");
scanf("%30[^\n]%*c", givenName);
printf("Enter surname     : ");
scanf("%30[^\n]%*c", surname);
strcpy(fullName, givenName);
strcat(fullName, " ");
strcat(fullName, surname);
printf("Full name is %s\n", fullName);
```

We choose 62 here to accommodate 30+30 characters plus a blank space separator plus the null byte terminator.

Arrays of Strings

- Declare an array of strings
  ```
  char identifier[NO_OF_STRINGS][NO_OF_BYTES_PER_STRING];
  ```
- Create an array of 5 names, where each name holds up 30 characters.
  ```
  char names[5][31];
  ```
- Declare an array with an initialization:
  ```
  char names[5][31] = {"Harry", "Jean", "Jessica", "Irene", "Jim"};
  ```

Input and Output

- The address of a string in an array of string is `identifier[string index]`
- Example

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

#define NUM_NAMES 3

int main ()
{
    char names[NUM_NAMES][31];
    int i;

    for (i = 0; i < NUM_NAMES; i++)
    {
        printf("Enter the name %d: ", i+1);
        scanf("%30[^\n]%*c", names[i]);
    }

    printf("The names and their lengths:\n");
    for (i = 0; i < NUM_NAMES; i++)
        printf("%s %ld\n", names[i], strlen(names[i]));

    return 0;
}
```

Passing to Functions

- To pass an array of strings to a function, we specify the array name as the argument. To receive an array of strings as a function parameter, we use the form

```
dataType function_name (dataType identifier[][NO_OF_BYTES_PER_STRING], ...)
```

- We must specify the second dimension of the array.
- For example,

```
void foo (name[][31], int a)
{
    ...
}

foo (name, n);
```

Sorting an Array of Names

- Lets write a program that accepts several names and displays them in alphabetic order.

```
// bubbleSort sorts the elements of a[size] in ascending order
 void bubble(char a[][MC+1], int size)
 {
     int i, j;
     char temp[MC+1];

     for (i = size - 1; i > 0; i--)
     {
         for (j = 0; j < i; j++) {
             if (strcmp(a[j],a[j+1]) > 0)
             {
                 strcpy(temp, a[j]);
                 strcpy(a[j], a[j+1]);
                 strcpy(a[j+1], temp);
             }
         }
     }
 }
```

- See section 3 of module F (seneca website) for the whole program.

Summary

- String Input
  - scanf
  - gets
  - using getchar()
- String Functions and Arrays of Strings
  - Functions
    - strlen
    - strcpy
    - strcmp
    - strcat
  - Arrays of Strings
    - Input and Output
    - Passing to Functions
    - Sorting an Array of Names

Q&A

In-Class Practice

- Write a function with the header

  ```
  int removeDuplicates(char name[][31], int age[], int *n)
  ```

  that removes all duplicate records from a table. The table consists of *n records and each record consists of a null-terminated string containing a person's name and an int containing their age. *n holds the number of records in the table when removeDuplicates is called and the number of records in the updated table upon return from removeDuplicates. A duplicate recor is a record that has the same name as another record, but not necessarily the same age. When removing a duplicate record, your function keeps the record with the lower index and discards the record with the higher index. Your function returns the total number of records removed.

```
Before call to removeDuplicates
name[ ][]   age[ ]    *n
------------------------------
Homer          45      6
Marge          22
Homer          42
Bart           14
Marge          21
Lisa            8
```

```
After call to removeDuplicates
name[ ][]   age[ ]    *n
----------------------
Homer          45      4
Marge          22
Bart           14
Lisa            8
```