Strings and Characters

Fundamentals of Programming

Lecture 5

Contents

- Character arrays
- Character pointers
- Read and write strings
- Library functions string.h

Character arrays

Character array

- A character array can store individual ASCII characters or a string of characters
- Last character in a char array is the NULL character, which is added automatically
- If the array is initialised, the data need not fill every space in the array

```
char a[3] = "";
char letters[6] = {'H','e','l','l','o'};
char message[20] = "Good morning!";
```

Elements in an array

```
char msg[6] = "Hello";
```

Index	**0**	**1**	**2**	**3**	**4**	**5**
Element	`msg[0]`	`msg[1]`	`msg[2]`	`msg[3]`	`msg[4]`	
Data	`H`	`e`	`l`	`l`	`o`	`\0`

Storing string data

- When using SCANF() to read string data, the %s format specifier must be used
- Arrays specified as the destination storage location in SCANF() does not need the ampersand (&) symbol when %s is used
- Remember that the string data must not contain any blank spaces

Char array example 1

- FOR loop is used to read one character at a time
- String data is stored in word[]
- SCANF() uses %c
- PRINTF() uses %s to output data in word[]

Avoid input using this loop

```
#include <stdio.h>
int main()
{
  char word[6];
  int n;
  printf("Enter word: ");
  for (n = 0; n < 6; n++) {
    scanf("%c", &word[n]);
  }
  printf("%s\n", word);
  return 0;
}
```

Char array example 2

- No loop used
- String data is stored in word[]
- SCANF() uses %s
- PRINTF() uses %s to output data in word[]

```
#include <stdio.h>
int main()
{
  char word[6];
  printf("Enter word: ");
  scanf("%s", word);
  printf("%s\n", word);
  return 0;
}
```

Character pointers

Character pointer

- A character pointer may be declared and initialised with string data
- Use %c format specifier for char
- Use %s format specifier for string
- Multiple character pointers is used to stitch together a longer text

```
char *word = "Hello";
printf("%s", word);
```

Character pointer example 1

```
#include <stdio.h>
int main()
{
  char *word = "Hello";
  printf("%s\n", word);
  return 0;
}
```

Character pointer example 2

```
#include <stdio.h>
int main()
{
  char *word1 = "Hello";
  char *word2 = "World";
  printf("%s %s\n", word1, word2);
  return 0;
}
```

Read and write strings

Puts() and Fputs()

- PUTS() function is used to output string data to the terminal screen
- String data is from a char array, or a char pointer to char array

```
puts(*<string>)
```

- FPUTS() function is used to output string data to a stream
- Stream can be a file pointer, char array, char pointer

```
fputs(*<string>,*<stream>)
```

Puts() example 1

- PUTS() can output string data directly to the terminal

```
#include <stdio.h>
int main()
{
  puts("Hello");
  return 0;
}
```

Puts() example 2

- PUTS() can output string data from a char array to the terminal

```
#include <stdio.h>
int main()
{
  char word[20] = "Hello";
  puts(word);
  return 0;
}
```

Puts() example 3

- PUTS() can output string data from a char pointer to the terminal

```
#include <stdio.h>
int main()
{
  char *word = "Hello";
  puts(word);
  return 0;
}
```

Fputs() example 1

- FPUTS() output string data directly to terminal screen
- Output stdout is the terminal screen

```
#include <stdio.h>
int main()
{
   fputs("Hello\n", stdout);
   return 0;
}
```

Fputs() example 2

- FPUTS() is used to output string data to terminal screen via character pointer
- Character pointer must be initialised with string data
- Output stdout is the terminal screen

```
#include <stdio.h>
int main()
{
  char *word = "Hello";
  fputs(word, stdout);
  return 0;
}
```

Gets() and Fgets()

- GETS() function is used to read string data
- Buffer size is not fixed, it is unsafe (with infinite characters as input)
- Data storage is a char array, or a char pointer to a char array

```
gets(*<string>)
```

- FGETS() is recommended because of fixed buffer size
- Size refers to number of characters to be read in bytes
- Stream is stdin to get input values from keyboard

```
fgets(*<string>,<size>,*<stream>)
```

Gets() example 1

- GETS() is used to read string data
- String data is stored in a char array
- Size of the string data is limited by the size of the char array

Not recommend to use gets(), outdated

```
#include <stdio.h>
int main()
{
   char word[20];
   printf("Enter word: ");
   gets(word);
   printf("%s\n", word);
   return 0;
}
```

Gets() example 2

- GETS() is used to read string data
- String data is referenced via a char pointer
- Size of the string data has no limitations

Not recommend to use gets(), outdated

```
#include <stdio.h>
int main()
{
  char word[20];
  char *p;
  p = &word[0];
  printf("Enter word: ");
  gets(p);
  printf("%s\n", word);
  return 0;
}
```

Fgets() example

- FGETS() is used to read string data in a controlled manner, since buffer size is fixed
- Buffer configured for char data
- Buffer size is N−1, where N = 20

```
#include <stdio.h>
int main()
{
  char word[20];
  char *p;
  p = &word[0];
  printf("Enter word: ");
  fgets(p, 20*sizeof(char), stdin);
  printf("%s\n", word);
  return 0;
}
```

Library functions
string.h

String.h library

- The string.h library has functions to edit character arrays
- Most functions returns a modified string, but some are values
- Character array size must be set before use in the program

String.h functions

- strcpy() to copy strings
- strcat() to concatenate two short strings into a longer string
- strcmp() to compare two strings
- strlen() to determine string length

Strcpy()

- STRCPY() is used to copy source (src) string to destination (dest) string

```
strcpy(dest,src)
```

- Example:

```
char x[20] = "Operating";
char y[20] = "System";
strcpy(x,y);
// x = System
// y = System
```

Strcpy() example

```
#include <stdio.h>
#include <string.h>
int main()
{
  char x[20] = "Operating";
  char y[20] = "System";
  printf("x = %s, y = %s\n", x, y);
  strcpy(x,y);
  printf("x = %s, y = %s\n", x, y);
  return 0;
}
```

Strcat()

- STRCAT() is used to concatenate two strings (src and dest) together, and save the new data in the destination (dest)

```
strcat(dest,src)
```

- Example:

```
char x[20] = "Operating";
char y[20] = "System";
strcat(x,y);
// x = OperatingSystem
// y = System
```

combine x and y, and save to x (y not changed)

Strcat() example

```
#include <stdio.h>
#include <string.h>
int main()
{
  char x[20] = "Operating";
  char y[20] = "System";
  printf("x = %s, y = %s\n", x, y);
  strcat(x,y);
  printf("x = %s, y = %s\n", x, y);
  return 0;
}
```

Strcmp()

- STRCMP() is used to compare between two strings
- Returns an integer value that indicates similarity

```
strcmp(x,y)
```

- Example:

```
int i;
char *x = "123";
char *y = "123";
i = strcmp(x,y);  // i = 0
```

Strcmp() return value

- STRCMP() compares x against y by taking ASCII values in each character array, x[] and y[], and determines the difference
- The return value is the differences
- If x > y in terms of ASCII values, then return value > 0
- If x < y in terms of ASCII values, then return value < 0
- If x == y in terms ASCII values, return value is 0

The return value is compiler specific, the positive/negative/zero are the same, but the absolute value may vary based on compiler.

Strcmp() example 1

- Character arrays x[] and y[] are declared and initialised with words
- Using STRCMP(), the x[] is compared against y[]

-1

- The return value (i = –4) is negative because the word in y[] contains larger ASCII values, even though the word in x[] has more letters

```
#include <stdio.h>
#include <string.h>
int main()
{
  int i;
  char x[20] = "Operating";
  char y[20] = "System";
  i = strcmp(x,y);
  printf("i = %d\n", i);
  return 0;
}
```

Strcmp() example 2

- Letter “c” has ASCII value 99
- Letter “a” has ASCII value 97
- The return value (i = $\overset{1}{2}$) is the difference of “c” – “a” or 99 – 97
- In this example, return value is positive

either 1 (positive), -1 (negative) or 0 (equal)

```
#include <stdio.h>
#include <string.h>
int main()
{
    int i;
    char x[20] = "c"; // 99
    char y[20] = "a"; // 97                1
    i = strcmp(x,y); // 99 - 97 = 2
    printf("i = %d\n", i);               > 0
    return 0;
}
```

Strcmp() example 3

- The return value (i = –1) is the difference of “ab” – “ac”

- In this example, return value is negative

```
#include <stdio.h>
#include <string.h>
int main()
{
    int i;
    char x[20] = "ab"; //
    char y[20] = "ac"; //
    i = strcmp(x,y); //
    printf("i = %d\n", i);
    return 0;
}
```

Strlen()

- STRLEN() is used to determine the length of a string text
- Returns an integer value that indicates the string length

```
strlen(x)
```

- Example:

```
int i;
char x[20] = "Hello";
i = strlen(x); // i = 5
```

Strlen() example 1

- The STRLEN() will return a value of 5, because there are five letters in “Hello”
- STRLEN() requires an integer variable to store the return value
- Variable i will have a value of 5

```
#include <stdio.h>
#include <string.h>
int main()
{
  int i;
  char x[20] = "Hello";
  i = strlen(x);
  printf("i = %d\n", i);
  return 0;
}
```

Strlen() example 2

- Any ASCII character in the character array will be counted, including blankspaces
- Using STRLEN() on the text “Hello World” will return a value of 11

```
#include <stdio.h>
#include <string.h>
int main()
{
  int i;
  char x[20] = "Hello World";
  i = strlen(x);
  printf("i = %d\n", i);
  return 0;
}
```

Functions with string arrays

String array as input parameter

- Functions can take in any data type as input parameters
- String array can be processed as a whole
- String array elements can be processed individually
- When using string arrays or string pointers as input parameters, use STRLEN() for better control of the FOR and WHILE loops

Function example 1

- String array word[] is an input parameter to display()
- Function display() will print on character then two blank spaces

```
#include <stdio.h>
#include <string.h>
void display(char x[])
{
  int n;
  for (n=0; n<strlen(x); n++) {
    printf("%c  ", x[n]);
  }
  printf("\n");
}

int main()
{
  char word[10] = "Hello";
  display(word);
  return 0;
}
```

Function example 2

- String pointer word is an input parameter to display()
- Function display() will print the same word depending on the number of characters in the string data

```
#include <stdio.h>
#include <string.h>
void display(char *x)
{
  int n;
//for (n=0; n<strlen(x); n++) {
    printf("%s ", x);
//}
  printf("\n");
}

int main()
{
  char *word = "Hello";
  display(word);
  return 0;
}
```

Summary

- Character array is accessed via character pointer
- Character arrays are assigned with a string or single ASCII characters
- Character pointer can easily process 2-dimensional character arrays
- Use FPUTS() and FGETS() instead of PUTS() and GETS()
- String.h library file provide more complex functionality when operating on strings

Further readings

- C: How to Program, 8th Edition, Paul Deitel & Harvey Deitel
  - Characters and strings: pp. 366–368
  - Character arrays: pp. 258–260
- The C Programming Language, 2nd Edition, Kernighan & Ritchie
  - Strings and characters: pp. 18–22
  - Character arrays: pp. 29–30
  - Character pointers: pp. 93–95