C String (1)

Young W. Lim

December 9, 2017
### 0.1 A constant character string

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

int main(void) {
    char *s = "Hello, World!";
    int i, len;

    printf("length: %u \n", (unsigned) strlen(s));
    len = strlen(s);

    for (i=0; i<len; ++i) {
        printf("*(s+%d)= %c \n", i, *(s+i));
    }

    // s[5] = 0; // segmentation error
}
```

---

**t2.out**

```
length: 13
*(s+0)= H
*(s+1)= e
*(s+2)= l
*(s+3)= l
*(s+4)= o
*(s+5)= ,
*(s+6)=
*(s+7)= W
*(s+8)= o
*(s+9)= r
*(s+10)= l
*(s+11)= d
*(s+12)= !
```
\*(s+3)= 1
\*(s+4)= 0
\*(s+5)= ,
\*(s+6)= '
\*(s+7)= \'
\*(s+8)= 0
\*(s+9)= r
\*(s+10)= l
\*(s+11)= d
\*(s+12)= !
Segmentation fault

the pointer notation

- s is the address of a memory location where the 1st element is stored.
- s+2 is the address of a memory location where the 3rd element is stored.
- \*(s+2) denotes therefore the 3rd element.

the subscript notation

- \*(s+2) is the same as s[2]
- though we can use s[2], no array elements are allocated.
- in the memory, char *s allocates only single character pointer.
- no array of characters is allocated.

changing a constant character string

- char *s = "Hello, World!";
- character pointer s is declared with a initialization.
- the content of s is an address where a character can be.
- "Hello, World!" is a constant character string
- it is stored in the read-only memory location (predefined by compiler).
- "Hello, World!" returns the 1st address (the address of 'H')
- s points to this address.
- though this string is a constant but is not explicitly declared with const.
- therefore, no error message will be shown.
- but, if we execute, the "Segmentation fault" error will occur.
- this is because s[5]=0 attempts to change its element in the read-only memory location.
- we can compile but cannot execute normally.
0.2 The null terminating character

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

int main(void) {
    char s[100] = "Hello, World!";
    int i, len;

    printf("length: %u \n", (unsigned) strlen(s));
    len = strlen(s);

    for (i=0; i<len; ++i) {
        printf("*(s+%d)= %c \n", i, *(s+i));
    }

    printf("s = %s \n", s);
    s[5] = 0;
    printf("s = %s \n", s);

    for (i=0; i<len; ++i) {
        printf("*(s+%d)= %c \n", i, *(s+i));
    }

    printf("s[5] = %c %d %x \n", s[5], s[5], s[5]);
    printf("s[6] = %c %d %x \n", s[6], s[6], s[6]);
}
```

Output:
```
length: 13
*(s+0)= H
*(s+1)= e
*(s+2)= l
*(s+3)= l
*(s+4)= o
*(s+5)= ,
*(s+6)=
*(s+7)= W
*(s+8)= o
*(s+9)= r
*(s+10)= l
*(s+11)= d
*(s+12)= !
s= Hello, World!
```
Hello
*(s+0)= H
*(s+1)= e
*(s+2)= l
*(s+3)= l
*(s+4)= o
*(s+5)=
*(s+6)=
*(s+7)= W
*(s+8)= o
*(s+9)= r
*(s+10)= l
*(s+11)= d
*(s+12)= !
s[5] = 32 20

printf(“s= %s \n”, s);

• %s prints characters whose starting address is given by s.
• the end of characters is followed by 0 (null terminating character).
• char s[100] allocates 100 consecutive character locations in memory.
• s is the array name and the starting address.
• s[5]= 0 forces the last characters to be s[4].
• therefore s[0], s[1], s[2], s[3], s[4] will be printed.

0.3 Strings in a 2-dimensional array

h1.c

#include <stdio.h>
#include <string.h>
#define ROW 4
#define COL 10

int main(void) {
    char S2D[4][10] = { "Baker", "John", "Thomas", "Catherine"};
    int i, j;

    printf("-------------\n");
    for (i=0; i<ROW; ++i) {
        for (j=0; j<COL; ++j) {
            printf("%c", S2D[i][j]);
        }
        printf("\n");}
...
Strings stored in 2-dimensional array

- the string "Baker" is stored in the 1st row (the starting address is S2D[0])
- the string "John" is stored in the 2nd row (the starting address is S2D[1])
- the string "Thomas" is stored in the 3rd row (the starting address is S2D[2])
- the string "Catherine" is stored in the 4th row (the starting address is S2D[3])
- S2D takes 40 bytes (= 4 · 10 · 1)
- null terminating character '\0' is stored as 0x0
- when there are less initializer than the number of element, the array elements are initialized with the given initializers first and the remaining elements with zero.
- cannot use the assign statement to assign a string to an array
- S2D[0] = "Stuart"; does not working
- can assign characters to an array individually
  
  ```
  S2D[0][0] = 'S';
  S2D[0][1] = 't';
  S2D[0][2] = 'u';
  S2D[0][3] = 'a';
  S2D[0][4] = 'r';
  S2D[0][5] = 't';
  S2D[0][6] = '\0';
  ```

- can use the string copy function defined in `<string.h>`

  ```
  strcpy(S2D[0], "Stuart");
  ```
0.4 Strings in a 1-dimensional array

h2.c

```
#include <stdio.h>
#define ROW 4
#define COL 10

int main(void) {
    int i, j;

    printf("------------*(SP[i]+j)------------\n");
    for (i=0; i<ROW; ++i) {
        for (j=0; j<COL; ++j) {
            printf("%2c ", *(SP[i]+j));
        }
        printf("\n");
    }

    printf("------------*(SP[i]+j)------------\n");
    for (i=0; i<ROW; ++i) {
        for (j=0; j<COL; ++j) {
            printf("%2x ", *(SP[i]+j));
        }
        printf("\n");
    }

    printf("------------SP[i][j]------------\n");
    for (i=0; i<ROW; ++i) {
        for (j=0; j<COL; ++j) {
            printf("%2c ", SP[i][j]);
        }
        printf("\n");
    }

    printf("------------SP[i][j]------------\n");
    for (i=0; i<ROW; ++i) {
        for (j=0; j<COL; ++j) {
            printf("%2x ", SP[i][j]);
        }
        printf("\n");
    }
```

Page 7
SP[0] = "Stuart";

printf("SP[0] = %s \n", SP[0]);

Strings stored in 1-dimensional array

- the 1st string "Baker"
- the 2nd string "John"
- the 3rd string "Thomas"
- the 4th string "Catherine"

  - all these strings are constant strings (elements cannot be changed)
  - stored in the read-only memory section
  - each returns the address of the first character (the staring address)
• SP[0] is the address of ‘B’
• SP[1] is the address of ‘J’
• SP[2] is the address of ‘T’
• SP[3] is the address of ‘C’
• SP[0]+1 is the address of ‘a’
• SP[1]+1 is the address of ‘o’
• SP[2]+1 is the address of ‘h’
• SP[3]+1 is the address of ‘a’

• *(SP[0]+i) is the same as SP[0][i]
• *(SP[1]+i) is the same as SP[1][i]
• *(SP[2]+i) is the same as SP[2][i]
• *(SP[3]+i) is the same as SP[3][i]

• SP is the 1-dimensional array name whose element is a character pointer (char *)
• SP can hold the address that are returned by "Stuart"
  SP[0] = "Stuart" is possible

• the null terminating character is denoted by *
  Baker * John
  John * Thomas
  Thomas * Cat
  Catherine *

• each null terminated string is stored one after the other without any space
• after the null terminating character of the given string, the first character of the next string is stored.
0.5 Displaying addresses of strings

```c
#include <stdio.h>
#define ROW 4
#define COL 10

int main(void) {
    int i, j;
    char *p;

    printf("------------SP[i]---------------\n");
    for (i=0; i<ROW; ++i) {
        printf("SP[%d]= %p \n", i, SP[i]);
    }
    printf("\n");

    printf("------------SP[i]+j-------------\n");
    for (i=0; i<ROW; ++i) {
        p = SP[i];
        j = 0;
        while (*p) {
            printf("(SP[%d]+%d)= %p \n", i, j, SP[i]+j);
            j++;
            p = SP[i]+j;
        }
        printf("(SP[%d]+%d)= %p \n", i, j, SP[i]+j);
    }
    printf("\n");

    printf("------------*(SP[i]+j)----------\n");
    for (i=0; i<ROW; ++i) {
        p = SP[i];
        j = 0;
        while (*p) {
            printf("*(SP[%d]+%d)= %c \n", i, j, *(SP[i]+j));
            j++;
            p = SP[i]+j;
        }
        printf("*(SP[%d]+%d)= %c \n", i, j, *(SP[i]+j));
    }
    printf("\n");
}
```

h3.out

------------------SP[i]------------------

SP[0] = 0x4008d8
SP[1] = 0x4008de
SP[2] = 0x4008e3
SP[3] = 0x4008ea

------------------SP[i]+j------------------

(SP[0]+0) = 0x4008d8
(SP[0]+1) = 0x4008d9
(SP[0]+2) = 0x4008da
(SP[0]+3) = 0x4008db
(SP[0]+4) = 0x4008dc
(SP[0]+5) = 0x4008dd
(SP[1]+0) = 0x4008de
(SP[1]+1) = 0x4008df
(SP[1]+2) = 0x4008e0
(SP[1]+3) = 0x4008e1
(SP[1]+4) = 0x4008e2
(SP[2]+0) = 0x4008e3
(SP[2]+1) = 0x4008e4
(SP[2]+2) = 0x4008e5
(SP[2]+3) = 0x4008e6
(SP[2]+4) = 0x4008e7
(SP[2]+5) = 0x4008e8
(SP[2]+6) = 0x4008e9
(SP[3]+0) = 0x4008ea
(SP[3]+1) = 0x4008eb
(SP[3]+2) = 0x4008ec
(SP[3]+3) = 0x4008ed
(SP[3]+4) = 0x4008ee
(SP[3]+5) = 0x4008ef
(SP[3]+6) = 0x4008f0
(SP[3]+7) = 0x4008f1
(SP[3]+8) = 0x4008f2
(SP[3]+9) = 0x4008f3

----------------*(SP[i]+j)----------------

*(SP[0]+0) = B
*(SP[0]+1) = a
*(SP[0]+2) = k
*(SP[0]+3) = e
*(SP[0]+4) = r
*(SP[0]+5) =
*(SP[1]+0) = J
*(SP[1]+1) = o
*(SP[1]+2) = h
*(SP[1]+3) = n
*(SP[1]+4) =
Displaying the addresses and characters of the given four strings

- the code segment for displaying the addresses of the characters of the given string

```c
p = SP[i];
j = 0;
while (*p) {
    printf("(SP[%d]+%d)= %p \n", i, j, SP[i]+j);
    j++;
    p = SP[i]+j;
}
```

- SP[i] is the starting address of the (i+1)-th string
- j is the position index to each character in the given string
- p = SP[i]+j is the address of the (j+1)-th characters in the (i+1)-th string
- *p becomes zero, when p points to the null terminating character

- `strlen("Baker")` → 5 + 1 = 6
- `strlen("John")` → 4 + 1 = 5
- `strlen("Thomas")` → 6 + 1 = 7
- `strlen("Catherine")` → 9 + 1 = 10

- total 28 bytes for string constants and 4 · 8 = 32 bytes for the character pointer 1-dimensional array
- total 40 bytes for the 2-dimensional array