Topic 6. Character Arrays and Strings in C

COMP ENG 2SH4

Principles of Programming

McMaster University, 2015

Instructor: Sorina Dumitrescu

Type **char**

- Size: 1 byte memory (8 bits)
- Variables and constants of type **char** have integer values.
- Range of values is machine-dependent:

  -128 ~ 127 or 0 ~ 255
- The C standard defines the **basic source** and the **basic execution character sets** as consisting of
  - The 26 uppercase letters of the Latin alphabet
  - The 26 lowercase letters of the Latin alphabet
  - The 10 decimal digits
  - 29 graphic characters: !”#%&’()*+,-./:;<=>?[\]^_{\}~
  - Some control sequences

Type **char**

- The members of the basic source and executable character sets have to be representable with type **char** and must have **non-negative** values.

- American Standard Code for Information Interchange (**ASCII**) assigns values in 0 ~ 127 to a set of 128 characters including the basic characters specified above.

Character Constants

- A character constant is a single character written within single quotes. Ex: ‘X’.
- Its value is an **integer**.
  - ASCII: value of 'A' is 65, value of 'a' is 97.
- Some characters are written as escape sequences:
  - \n (newline), \t (tab)
  - corresponding character constants: '\n', '\t’

String Constants

- A **string constant** (or **string literal**) is written as a sequence of characters within double quotes.
  - “Hello“, “12_gh; *”
- A string literal is represented in memory as an array of characters, ending with the **null character** ‘\0’.
  - The null character has **integer value 0**; ‘\0’ is different from the digit zero (‘0’), and from the space character (‘ ‘)
- “Hello” is the array with elements: **‘H’, ‘e’, ‘l’, ‘l’, ‘o’, ‘\0’**
- A character constant and the string constant that contains only that character, are different !
  - “a” is an array with 2 elements: ‘a’ and ‘\0’.

Character Arrays. Strings

- A **character array** is an array whose elements are characters.

```
char ch_array[  ] = { ‘H', ‘e', ‘l', ‘l', ‘o’};
```

A **string** is a character array ending with the null character.

```
char my_string1[  ] = { ‘H', ‘e', ‘l', ‘l', ‘o', ‘\0' };
```

- Both **ch_array** and **my_string1** are character arrays.
- Only **my_string1** is a string.
- Character arrays are a data type.
- In C strings are not a data type.
- **Strings are a convention.**

Size of a String

- The **null character** is different from the characters we use in text messages.
- This is why it is used as a **termination symbol**.
- The null character **marks the end of the character array which represents the string.**
- The **size of a string** is the number of characters in the string before the termination symbol '\0'.

Defining and Initializing Strings

```
char my_string1[  ] = { ‘H', ‘e', ‘l', ‘l', ‘o', '\0' };
```

```
char my_string2[  ] = "Hello";
```

- Each of the above definitions allocates an **array of 6 chars** and initializes its elements to:

  `‘H', ‘e', ‘l', ‘l', ‘o', '\0‘`
- The **size of each string is 5.**
- The **size of each array is 6.**

Looping through a String

- We can use a **for** or **while** loop to visit and process all the characters in a string.
- The loop stops when the **null character** is encountered.

```
//incomplete code
#include <stdio.h>
int main(void){
    char b[ ]="blue";
    int i;
    for( i=0; ?? ; i++ ){
      // process current character b[i]
    } // end of loop
    return 0;
} // end of main
```

Looping through a String

- We can use a **for** or **while** loop to visit and process all the characters in a string.
- The loop stops when the **null character** is encountered.

```
//incomplete code
#include <stdio.h>
int main(void){
    char b[ ]="blue";
    int i;
    for( i=0; b[i]!='\0'; i++ ){
      // process current character b[i]
    } // end of loop
    return 0;
} // end of main
```

Looping through a String

- We can use a **for** or **while** loop to visit and process all the characters in a string.
- The loop stops when the **null character** is encountered.

```
//incomplete code
#include <stdio.h>
int main(void){
    char b[ ]="blue";
    int i=0;
    while(b[i]!='\0'){
      // process current character b[i]
            i++;
    } // end of loop
    return 0;
} // end of main
```

Computing the Size of a String

```
// incomplete code
#include <stdio.h>
int main(void){
    char b[ ]="blue";
    int i;
    int size=0;
    for( i=0; b[i]!='\0'; i++ ){
      // process current character b[i]
    } // end of loop
    return 0;
} // end of main
```

Computing the Size of a String

```
// complete code
#include <stdio.h>
int main(void){
    char b[ ]="blue";
    int i;
    int size=0;
    for( i=0; b[i]!='\0'; i++ ){
        size++;
    } // end of loop
    printf("The string has %d characters\n", size);
    return 0;
} // end of main
```

Computing the Size of a String

```
// complete code
#include <stdio.h>
int main(void){
    char b[ ]="blue";
    int i;
    //another way to compute the size
    int size=0;
    while( b[size]!='\0' )
        size++;

    printf("The string has %d characters\n", size);
    return 0;
} // end of main
```

Looping through a String

- Find the number of occurrences of letter ‘c’ or ‘C’ in a string.

```
//incomplete code
#include <stdio.h>
int main(void){
    char aa[ ]=“Computer science is fun!”;
    int i=0; // to loop through string
    int counter=0; // to count occurrences

    /*
      loop through string and do the work
    */
    printf(“%d\n”, counter); return 0;
} /* end of main */
```

Looping through a String

- Find the number of occurrences of letter ‘c’ or ‘C’ in a string.

```
//incomplete code
#include <stdio.h>
int main(void){
    char aa[ ]=“Computer science is fun!”;
    int i=0; // to loop through string
    int counter=0; // to count occurrences
    while( aa[i]!=‘\0’ ){
      //process current character aa[i]
      i++;
    } /* end of loop */
    printf(“%d\n”, counter); return 0;
} /* end of main */
```

Looping through a String

- Find the number of occurrences of letter ‘c’ or ‘C’ in a string.

```
//incomplete code
#include <stdio.h>
int main(void){
    char aa[ ]=“Computer science is fun!”;
    int i=0; // to loop through string
    int counter=0; // to count occurrences
    while( aa[i]!=‘\0’ ){
      //if aa[i] is ‘c’ or ‘C’ increment counter
      i++;
    } /* end of loop */
    printf(“%d\n”, counter); return 0;
} /* end of main */
```

Looping through a String

- Find the number of occurrences of letter ‘c’ or ‘C’ in a string.

```
#include <stdio.h>
int main(void){
    char aa[ ]=“Computer science is fun!”;
    int i=0; // to loop through string
    int counter=0; // to count occurrences
    while( aa[i]!=‘\0’ ){
      if( aa[i]==‘c’ || aa[i]==‘C’ )
          counter++;
      i++;
    } /* end of loop */
    printf(“%d\n”, counter); return 0;
} /* end of main */
```

Passing a String to a Function

- When passing a string to a function it is enough to **pass the name of the string** (i.e. the name of the char array).
- The function **can access and modify** the characters in the string.
- It is **not necessary to pass the size** of the char array.
- The function knows that **the null character marks the end of the string.**

```
void reverse_string( char x[] ) {



    /* find the size of the string */



    /* reverse the string */



} // end of function
```

Passing a String to a Function

- When passing a string to a function it is enough to **pass the name of the string** (i.e. the name of the char array).
- The function **can access and modify** the characters in the string.
- It is **not necessary to pass the size** of the char array.
- The function knows that **the null character marks the end of the string.**

```
//incomplete code
void reverse_string( char x[] ) {
    int i, size=0, temp;
    /* find the size of the string */
    while( x[size] != '\0' ) /* ( x[size] != 0 )  */
        size++;

    /* reverse the string */



} //end of function
```

Passing a String to a Function

- When passing a string to a function it is enough to **pass the name of the string** (i.e. the name of the char array).
- The function **can access and modify** the characters in the string.
- It is **not necessary to pass the size** of the char array.
- The function knows that **the null character marks the end of the string.**

```
void reverse_string( char x[] ) {
    int i, size=0, temp;
    /* find the size of the string */
    while( x[size] != '\0' )
        size++;
/* Attention: size does not include '\0' in the count */
    for(i=0; i< size/2; i++){
      temp=x[i];
      x[i]=x[size-1-i];
      x[size-1-i]=temp; } /* end of for loop */
} // end of function
```

Console String I/O

- A string can be input/output with **scanf/printf** and conversion specifier **%s**.
- The string has to be read into a **char** array large enough to hold the whole string including the terminating '\0'. This array must already exist in memory.
- With **scanf** the name of the array is used without **&**
- To specify that at most ***m*** characters are to be read, use

%ms

```
char aa[ 20 ], bb[ 10 ];
printf( "Enter a string: " );
scanf( "%s", aa ); /* reads seq. of characters until white
    space, and places them in array aa; adds '\0' at the
    end.
    CAUTION: scanf can write beyond the end of array aa */
scanf( "%5s", bb ); /* reads a seq. of at most 5
    characters and stores them in array bb; adds '\0' at
    the end */
```

Console String I/O

```
char my_string[8];
… /* some code */
printf( "%s\n", my_string);
```

- **printf** with **%s** prints the characters in the **char** array until the first occurrence of the null character.
- The two following pieces of code have the same effect.

```
printf( "%s\n", my_string);
```

```
for( i = 0; my_string[i] != '\0'; i++)
   printf( “%c", my_string[i]);
printf(“\n”);
```

Console String I/O

```
char string[ 20 ] = "Canada";
printf( "string is: %s\n", string);
string[3] = 0;     /* string[3]='\0' */
printf( "string is: %s\n", string);
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
‘C’	‘a’	‘n’	‘a’	‘d’	‘a’	‘\0’													
‘C’	‘a’	‘n’	‘\0’	‘d’	‘a’	‘\0’													

string is: Canada
string is: Can

Console String I/O

- **gets(char array):** **reads the next input line** into the character array. Replaces the terminating newline with '\0'.

- **puts(string):** prints the characters in the string followed by \n, on the screen.

```
#include <stdio.h>
int main(void){
    char a[20];
    puts("Please input five characters on a line.");
    gets(a);
    printf("a[4]=%c, its integer value is %d\n", a[4], a[4]);
    printf("a[5]=%c, its integer value is %d\n", a[5], a[5]);
    puts(a);
    return 0;}
```

Optional: Formatted String Output

- With **printf** the **output field width and precision** can be specified:
- Field width specification is taken into account only if it indicates a field larger than the number of characters in the string.
- Ex: %15s
- Precision indicates the number of characters to be output.
- Ex: %.2s:
- %10.6s (both field width and precision)

```
char a_string[]="Canada";
printf("%s\n", a_string );
printf("%4s\n", a_string );
printf("%12s\n", a_string );
printf("%.4s\n", a_string );
printf("%12.4s\n", a_string );
printf("%-12.4s\n", a_string );
```

```
Canada
Canada
      Canada
Cana
        Cana
Cana
Press any key to continue_
```

Character-Handling Library

- Functions to manipulate character data
- #include<ctype.h>
- int isdigit(int c) – determines if c is a digit
- int isalpha(int c) – determines if c is a letter
- int islower(int c); int isupper(int c) – determines if c is a lowercase letter, uppercase letter, resp.
- int toupper(int c); int tolower(int c) – converts to uppercase, lowercase respectively.
- More examples in Fig. 1, pp. 349,350 in textbook (Chap. 8)

Character-Handling Library

- Functions to manipulate character data
- #include<ctype.h>
- int isdigit(int c) – determines if c is a digit
- int isalpha(int c) – determines if c is a letter
- int islower(int c) – determines if c is a lowercase letter
- int isupper(int c) – determines if c is an uppercase letter
- int isxdigit(int c) – determines if c is a hexadecimal digit (A-F,a-f,0-9)
- More examples in Fig. 1, pp. 349,350 in textbook (Chap. 8)

String-Handling Library

- Functions to manipulate string data:
  - Copying and concatenating strings
  - Comparing strings
  - Determining the length of a string (**strlen**)
  - Searching strings for characters and other strings
- #include<string.h>

Other Standard I/O Functions

- sscanf() - works like scanf, but reads from an array of characters, rather than from the keyboard
- sscanf(my_array, “%c”, &var)
- sprintf() - like printf, except that the outpu is stored in an array of characters instead of printed on the screen.
- sprintf(my_array, “%c%, var)

- Textbook Reading:
- Chapter 8 (without pointers)
  - sections 8.1,2