UNIVERSITY of LOUISVILLE
dare to be great

C Chapter 8: Strings

CECS130
Introduction to Programming Languages
Dr. Roman V. Yampolskiy

Introduction

- Programs can process characters and strings.
- Used to make
  - Word processors
  - Page layout software
  - Typesetting programs

Strings

- Many languages include built-in support for Strings
- Typical examples include
  - Basic
  - Pascal
  - Scheme/Lisp
  - Java
- "Built-in" here implies behind the scenes details are managed by the compiler and/or the run-time environment

Strings in C

- There are no Strings in C.
- There is an agreed upon convention to store string like data structures in arrays of characters.
- Part of the convention includes terminating strings with a null character.
- In many cases "string" operations in C look just like those found in other languages.
  - Do not be deceived. C...strings must be managed properly (by you)!
- Character array - an array whose components are of type char
- String - a sequence of zero or more characters enclosed in double quote marks

C Strings (Character Arrays)

- There is a difference between `'A'` and `"A"`
  - `'A'` is the character `A`
  - `"A"` is the string `A`
- Because strings are null terminated, `"A"` represents two characters, `'A'` and `'\0'`
- Similarly, `"Hello"` contains six characters, `'H'`, `'e'`, `'l'`, `'l'`, `'o'`, and `'\0'`

Fundamentals of Strings and Characters

- Characters
  - An **`int`** value represented as a character in single quotes
  - **`'z'`** represents the integer value of **`z`**
- Strings
  - Series of characters treated as a single unit
    - Can include letters, digits and special characters (**`*`**, **`/`**, **`$`**)
  - String literal (string constant) - written in double quotes
    - **`"Hello"`**
  - Strings are arrays of characters
    - String a pointer to first character
    - Value of string is the address of first character

Character Handling Library

- Character handling library
  - Includes functions to perform useful tests and manipulations of **character** data
  - Each function receives a character (an **`int`**) or **`EOF`** as an argument
- The following slide contains a table of all the functions in **`<ctype.h>`**

Character Handling Library

Prototype	Description
`int isdigit( int c )`	Returns **`true`** if **`c`** is a digit and **`false`** otherwise.
`int isalpha( int c )`	Returns **`true`** if **`c`** is a letter and **`false`** otherwise.
`int isalnum( int c )`	Returns **`true`** if **`c`** is a digit or a letter and **`false`** otherwise.
`int isxdigit( int c )`	Returns **`true`** if **`c`** is a hexadecimal digit character and **`false`** otherwise.
`int islower( int c )`	Returns **`true`** if **`c`** is a lowercase letter and **`false`** otherwise.
`int isupper( int c )`	Returns **`true`** if **`c`** is an uppercase letter; **`false`** otherwise.
`int tolower( int c )`	If **`c`** is an uppercase letter, **`tolower`** returns **`c`** as a lowercase letter. Otherwise, **`tolower`** returns the argument unchanged.
`int toupper( int c )`	If **`c`** is a lowercase letter, **`toupper`** returns **`c`** as an uppercase letter. Otherwise, **`toupper`** returns the argument unchanged.
`int isspace( int c )`	Returns **`true`** if **`c`** is a white-space character—newline (**`'\n'`**), space (**`' '`**), form feed (**`'\f'`**), carriage return (**`'\r'`**), horizontal tab (**`'\t'`**), or vertical tab (**`'\v'`**)—and **`false`** otherwise
`int iscntrl( int c )`	Returns **`true`** if **`c`** is a control character and **`false`** otherwise.
`int ispunct( int c )`	Returns **`true`** if **`c`** is a printing character other than a space, a digit, or a letter and **`false`** otherwise.
`int isprint( int c )`	Returns **`true`** value if **`c`** is a printing character including space (**`' '`**) and **`false`** otherwise.
`int isgraph( int c )`	Returns **`true`** if **`c`** is a printing character other than space (**`' '`**) and **`false`** otherwise.

```c
   /* Using functions isdigit, isalpha, isalnum, and isxdigit */
#include <stdio.h>
#include <ctype.h>

int main()
{
   printf( "%s\n%s%s\n%s%s\n\n", "According to isdigit: ",
       isdigit( '8' ) ? "8 is a " : "8 is not a ", "digit",
       isdigit( '#' ) ? "# is a " :
       "# is not a ", "digit" );
   printf( "%s\n%s%s\n%s%s\n%s%s\n%s%s\n\n",
       "According to isalpha:",
       isalpha( 'A' ) ? "A is a " : "A is not a ", "letter",
       isalpha( 'b' ) ? "b is a " : "b is not a ", "letter",
       isalpha( '&' ) ? "& is a " : "& is not a ", "letter",
       isalpha( '4' ) ? "4 is a " :
       "4 is not a ", "letter" );
   printf( "%s\n%s%s\n%s%s\n%s%s\n\n",
       "According to isalnum:",
       isalnum( 'A' ) ? "A is a " : "A is not a ",
       "digit or a letter",
       isalnum( '8' ) ? "8 is a " : "8 is not a ",
       "digit or a letter",
       isalnum( '#' ) ? "# is a " : "# is not a ",
       "digit or a letter" );
   printf( "%s\n%s%s\n%s%s\n%s%s\n%s%s\n%s%s\n",
       "According to isxdigit:",
       isxdigit( 'F' ) ? "F is a " : "F is not a ",
       "hexadecimal digit",
       isxdigit( 'J' ) ? "J is a " : "J is not a ",
       "hexadecimal digit",
```

1. Load header

2. Perform tests

3. Print

```
33         isxdigit( '7' ) ? "7 is a " : "7 is not a ",
34         "hexadecimal digit",
35         isxdigit( '$' ) ? "$ is a " : "$ is not a ",
36         "hexadecimal digit",
37         isxdigit( 'f' ) ? "f is a " : "f is not a ",
38         "hexadecimal digit" );
39     return 0;
40 }
```

3. Print

```
According to isdigit:
8 is a digit
# is not a digit

According to isalpha:
A is a letter
b is a letter
& is not a letter
4 is not a letter

According to isalnum:
A is a digit or a letter
8 is a digit or a letter
# is not a digit or a letter

According to isxdigit:
F is a hexadecimal digit
J is not a hexadecimal digit
7 is a hexadecimal digit
$ is not a hexadecimal digit
f is a hexadecimal digit
```

Program Output

Character String Declaration

In C, we declare a **character string** like so:

```
char my_name[my_name_length+1];
```

Notice that a character string is declared **exactly like** a `char` array; in fact, it **is** a `char` array.

The only difference in the declaration is that the length of the array of `char` elements that represents the string is **one greater** than the length of the string.

Storing and Accessing Strings

- A pointer-based string in C is an array of characters ending in the null terminator ('\0'), which indicates where a string terminates in memory. An array can be accessed via a pointer.
- A string can also be accessed via a pointer, which points to the first character in the string.
- You can declare a string variable using an array or a pointer. For example, the following two declarations are both fine:

```
char city[7] = "Dallas";        // Option 1
char *pCity = "Dallas";         // Option 2
```

Pointer Syntax

You can access city or pCity using the array syntax or pointer syntax. For example:

```
char city[7] = "Dallas";        // Option 1
char *pCity = "Dallas";         // Option 2
```

```
city[1]
*(city + 1)
pCity[1]
*(pCity + 1)
```

each displays character a (the second element in the string).

C Strings (Character Arrays)

- Consider the statement

```
char name[16];
```

- Because C strings are null terminated and name has sixteen components
  - the largest string that can be stored in name is `15`
- If you store a string of length, say 10 in `name`
  - the first 11 components of `name` are used and the last 5 are left unused

C Strings (Character Arrays)

- The statement

```
char name[16] = "John";
```

declares a string variable `name` of length 16 and stores `"John"` in it

- The statement

```
char name[] = "John";
```

declares a string variable name of length 5 and stores `"John"` in it

Reading Strings

```
#include <stdio.h>
main(){
	char color[12] = {'\0'};

	printf("Enter your favorite color: ");
	scanf("%s", color);
	printf("You entered: %s\n", color);
	system("pause");
}
```

Printing Strings

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

int main () {
    const int my_name_length       = 16;
    const int program_success_code =  0;
    char my_name[my_name_length + 1];

    strcpy(my_name, "Roman Yampolskiy");
    printf("My name is %s.\n", my_name);
    system("pause");
    return program_success_code;
}
```

```
Output: My name is Roman Yampolskiy.
```

Printing a String via Array Index

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

int main ()
{ /* main */
    const int my_name_length       = 16;
    const int program_success_code =  0;
    char my_name[my_name_length + 1];
    int  index;

    strcpy(my_name, "Roman Yampolskiy");
    printf("My name is ");
    index = 0;
    while (my_name[index] != '\0') {
        printf("%c", my_name[index]);
        index++;
    }
    printf(".\n");
    system("pause");
    return program_success_code;
} /* main */
Output: My name is Roman Yampolskiy.
```

Useful Functions

Function	Effect
`strcpy(s1, s2)`	Copies the string `s2` into the string variable `s1` The length of `s1` should be at least as large as `s2`
`strcmp(s1, s2)`	Returns a value < `0` if `s1` is less than `s2` Returns `0` if `s1` and `s2` are the same Returns a value > `0` if `s1` is greater than `s2`
`strlen(s)`	Returns the length of the string `s`, excluding the null character

strcpy Example

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

int main ()
{
    const int my_name_length       = 16;
    const int program_success_code =  0;
    char my_name[my_name_length + 1];
    char my_name2[my_name_length + 1];

    strcpy(my_name, "Roman Yampolskiy");
    printf("My name is %s.\n", my_name);
    strcpy(my_name2, my_name);
    printf("My name is %s.\n", my_name2);
    system("pause");
    return program_success_code;
}
```

```
My name is Roman Yampolskiy.
My name is Roman Yampolskiy.
```

In a printf statement, the placeholder for a character string is: %s

The `strlen` Function

The C Standard Library function `strlen` returns the length of the string that is passed to it, **EXCLUDING THE STRING TERMINATOR**:

```
my_name_length = strlen(my_name);
```

strlen Function Example

```
#include <stdio.h>
#include <string.h>
int main ()
{ /* main */
    printf("strlen(Roman Yampolskiy) = %d\n",
    strlen("Roman Yampolskiy"));
    system("pause");
    return 0;
} /* main */
Output: strlen(“Roman Yampolskiy") = 16
```

Passing a String as a Function Argument

Passing a string to a function as an argument is just like passing any other kind of array argument:

```
int main ()
{ /* main */
    char my_name[my_name_length + 1];
    char* my_name2 = (char*)NULL;
    ...
    print_a_string(my_name);
    ...
    print_a_string(my_name2);
    ...
} /* main */

void print_a_string (char* the_string)
```

String Comparisons

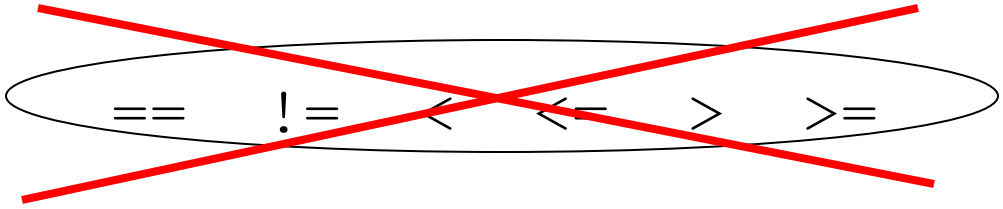
Just as numeric values can be compared, so can string values.

However, strings aren't scalars.

In C, two strings are defined to be equal if they have the exact same contents.

In C, strings are compared using the `strcmp` function from the C Standard Library.

The relational operators CANNOT be used to compare strings!



String Comparison

- C-strings are compared character by character
- If we are using the ASCII character set
  1. The string `"Air"` is smaller than the string `"Boat"`
  2. The string `"Air"` is smaller than the string `"An"`
  3. The string `"Bill"` is smaller than the string `"Billy"`
  4. The string `"Hello"` is smaller than `"hello"`

String Comparison is Case Sensitive

String comparison is **case sensitive**.

Thus, if two strings are identical, except that, in a single character, they differ by case – for example, an `"H"` for one string corresponds to an `"h"` for the other – then they will not be equal.

For example:

`"Henry"` is not equal to `"henry"`

String Comparison Example

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

int main () {

    const int my_name_length       = 12;   const int program_success_code =  0;
    char my_name[my_name_length + 1]; char my_name2[my_name_length + 1];
    char my_first_name[my_name_length + 1]; char my_first_name_lower[my_name_length + 1];
    char my_last_name[my_name_length + 1];

    strcpy(my_name, "Roman Yampolskiy");
    strcpy(my_name2, my_name);
    strcpy(my_first_name, "Roman");
    strcpy(my_first_name_lower, "roman");
    strcpy(my_last_name, "Yampolskiy");

    printf("strcmp(%s,%s) = %2d\n", my_name, my_name2, strcmp(my_name, my_name2));
    printf("strcmp(%s,%s) = %2d\n", my_first_name, my_first_name_lower, strcmp(my_first_name, my_first_name_lower));
    printf("strcmp(%s,%s) = %2d\n", my_last_name, my_first_name, strcmp(my_last_name, my_first_name));

    system("pause");
    return program_success_code;
}
```

String Comparison Example

```
strcmp(Roman Yampolskiy,Roman Yampolskiy) = 0
strcmp(Roman,roman) = -1
strcmp(Yampolskiy,Roman) = 1
```

Notice that the return value for `strcmp` can be interpreted as:

- **zero**: the strings are equal
- **negative**: the first string is less
- **positive**: the first string is greater

strcpy, strlen,strcmp: Examples

```
char studentName[21];
char myname[16];
char yourname[16];
```

Statement	Effect
`strcpy(myname, "John Robinson");`	`Myname = "John Robinson"`
`strlen("John Robinson");`	Returns 13, the length of the string "John Robinson"
`int len;` `len = strlen("Sunny Day");`	Stores 9 into `len`
`strcpy(yourname, "Lisa Miller");` `strcpy(studentName, yourname);`	`yourname = "Lisa Miller"` `studentName = "Lisa Miller"`
`strcmp("Bill", "Lisa");`	Returns a value < 0

`strcpy(yourname, "Kathy Brown");`
`strcpy(myname, "Mark G. Clark");`
`strcmp(myname, yourname);`
	`yourname = "Kathy Brown"`
	`myname = "Mark G. Clark"`
	Returns a value > 0

String Conversion Functions

- Conversion functions
  - In **<stdlib.h>** (general utilities library)
- Convert strings of digits to integer and floating-point values

Prototype	Description
`double atof( const char *nPtr )`	Converts the string **nPtr** to **double**.
`int atoi( const char *nPtr )`	Converts the string **nPtr** to **int**.
`long atol( const char *nPtr )`	Converts the string **nPtr** to long **int**.
`double strtod( const char *nPtr, char **endPtr )`	Converts the string **nPtr** to **double**.
`long strtol( const char *nPtr, char **endPtr, int base )`	Converts the string **nPtr** to **long**.
`unsigned long strtoul( const char *nPtr, char **endPtr, int base )`	Converts the string **nPtr** to **unsigned long**.

```c

   /* Using atof */
#include <stdio.h>
#include <stdlib.h>

int main()
{
   double d;

   d = atof( "99.0" );
   printf( "%s%.3f\n%s%.3f\n",
           "The string \"99.0\" converted to double is ", d,
           "The converted value divided by 2 is ",
           d / 2.0 );
   return 0;
}
```

```
The string "99.0" converted to double is 99.000
The converted value divided by 2 is 49.500
```

1. Initialize variable

2. Convert string

2.1 Assign to variable

3. Print

Program Output

Standard Input/Output Library Functions

- Functions in **<stdio.h>**
  - Used to manipulate character and string data

Function prototype	Function description
`int getchar( void );`	Inputs the next character from the standard input and returns it as an integer.
`char *gets( char *s );`	Inputs characters from the standard input into the array **s** until a newline or end-of-file character is encountered. A terminating null character is appended to the array.
`int putchar( int c );`	Prints the character stored in **c**.
`int puts( const char *s );`	Prints the string **s** followed by a newline character.
`int sprintf( char *s, const char *format, ... );`	Equivalent to **printf**, except the output is stored in the array **s** instead of printing it on the screen.
`int sscanf( char *s, const char *format, ... );`	Equivalent to **scanf**, except the input is read from the array **s** instead of reading it from the keyboard.

```
1
2     /* Using gets and putchar */
3  #include <stdio.h>
4
5  int main()
6  {
7     char sentence[ 80 ];
8     void reverse( const char * const );
9
10    printf( "Enter a line of text:\n" );
11    gets( sentence );
12
13    printf( "\nThe line printed backwards is:\n" );
14    reverse( sentence );
15
16    return 0;
17 }
18
19 void reverse( const char * const sPtr )
20 {
21    if ( sPtr[ 0 ] == '\0' )
22       return;
23    else {
24       reverse( &sPtr[ 1 ] );
25       putchar( sPtr[ 0 ] );
26    }
27 }
```

```
Enter a line of text:
Characters and Strings

The line printed backwards is:
sgnirtS dna sretcarahC
```

reverse calls itself using substrings of the original string. When it reaches the **'\0'** character it prints using **putchar**

1. Initialize variables

2. Input

3. Print

3.1 Function definition (note recursion)

String Manipulation Functions of the String Handling Library

- String handling library has functions to
  - Manipulate string data
  - Search strings
  - Tokenize strings
  - Determine string length

Function prototype	Function description
`char *strcpy( char *s1, const char *s2 )`	Copies string **s2** into array **s1**. The value of **s1** is returned.
`char *strncpy( char *s1, const char *s2, size_t n )`	Copies at most **n** characters of string **s2** into array **s1**. The value of **s1** is returned.
`char *strcat( char *s1, const char *s2 )`	Appends string **s2** to array **s1**. The first character of **s2** overwrites the terminating null character of **s1**. The value of **s1** is returned.
`char *strncat( char *s1, const char *s2, size_t n )`	Appends at most **n** characters of string **s2** to array **s1**. The first character of **s2** overwrites the terminating null character of **s1**. The value of **s1** is returned.

String Concatenation

Syntax:

```
#include <string.h>
char *strcat( char *str1, const char *str2 );
```

Description:

The strcat() function concatenates str2 onto the end of str1, and returns str1.

Example:

```
printf( "Enter your name: " );
scanf( "%s", name );
title = strcat( name, " the Great" );
printf( "Hello, %s\n", title );
```

```
1  /*
2     Using strcat and strncat */
3  #include <stdio.h>
4  #include <string.h>
5
6  int main()
7  {
8     char s1[ 20 ] = "Happy ";
9     char s2[] = "New Year ";
10    char s3[ 40 ] = "";
11
12    printf( "s1 = %s\ns2 = %s\n", s1, s2 );
13    printf( "strcat( s1, s2 ) = %s\n", strcat( s1, s2 ) );
14    printf( "strncat( s3, s1, 6 ) = %s\n", strncat( s3, s1, 6 ) );
15    printf( "strcat( s3, s1 ) = %s\n", strcat( s3, s1 ) );
16    return 0;
17 }
```

```
s1 = Happy
s2 = New Year
strcat( s1, s2 ) = Happy New Year
strncat( s3, s1, 6 ) = Happy
strcat( s3, s1 ) = Happy Happy New Year
```

1. Initialize variables

2. Function calls

3. Print

Search Functions of the String Handling Library

Function prototype	Function description
`char *strchr( const char *s, int c );`	Locates the first occurrence of character **c** in string **s**. If **c** is found, a pointer to **c** in **s** is returned. Otherwise, a **NULL** pointer is returned.
`size_t strcspn( const char *s1, const char *s2 );`	Determines and returns the length of the initial segment of string **s1** consisting of characters not contained in string **s2**.
`size_t strspn( const char *s1, const char *s2 );`	Determines and returns the length of the initial segment of string **s1** consisting only of characters contained in string **s2**.
`char *strpbrk( const char *s1, const char *s2 );`	Locates the first occurrence in string **s1** of any character in string **s2**. If a character from string **s2** is found, a pointer to the character in string **s1** is returned. Otherwise, a **NULL** pointer is returned.
`char *strrchr( const char *s, int c );`	Locates the last occurrence of **c** in string **s**. If **c** is found, a pointer to **c** in string **s** is returned. Otherwise, a **NULL** pointer is returned.
`char *strstr( const char *s1, const char *s2 );`	Locates the first occurrence in string **s1** of string **s2**. If the string is found, a pointer to the string in **s1** is returned. Otherwise, a **NULL** pointer is returned.
`char *strtok( char *s1, const char *s2 );`	A sequence of calls to **strtok** breaks string **s1** into "tokens"—logical pieces such as words in a line of text—separated by characters contained in string **s2**. The first call contains **s1** as the first argument, and subsequent calls to continue tokenizing the same string contain **NULL** as the first argument. A pointer to the current token is returned by each call. If there are no more tokens when the function is called, **NULL** is returned.

```c
/*
   Using strspn */
#include <stdio.h>
#include <string.h>

int main()
{
   const char *string1 = "The value is 3.14159";
   const char *string2 = "aehi lsTuv";

   printf( "%s%s\n%s%s\n\n%s\n%s%u\n",
           "string1 = ", string1, "string2 = ", string2,
           "The length of the initial segment of string1",
           "containing only characters from string2 = ",
           strspn( string1, string2 ) );
   return 0;
}
```

```
string1 = The value is 3.14159
string2 = aehi lsTuv

The length of the initial segment of string1
containing only characters from string2 = 13
```

1. Initialize variables

2. Function calls

3. Print

```c
/*
   Using strtok */
#include <stdio.h>
#include <string.h>

int main()
{
   char string[] = "This is a sentence with 7 tokens";
   char *tokenPtr;

   printf( "%s\n%s\n\n%s\n",
           "The string to be tokenized is:", string,
           "The tokens are:" );

   tokenPtr = strtok( string, " " );

   while ( tokenPtr != NULL ) {
      printf( "%s\n", tokenPtr );
      tokenPtr = strtok( NULL, " " );
   }

   return 0;
}
```

```
The string to be tokenized is:
This is a sentence with 7 tokens

The tokens are:
This
is
a
sentence
with
7
tokens
```

1. Initialize variables

2. Function calls

3. Print

String Searching

- **strstr()** will find a sub-string within a string.
- Library: string.h
- Prototype: char * strstr(const char *s1, const char *s2);
- Syntax:

```
char string1[]="red dwarf";
char string2[]="dwarf";
void *pointer;
pointer = strstr(string1, string2);
```

- strstr returns a pointer to the beginning of the sub-string or NULL if not found.

The End!