CSE1142 – Characters and Strings in C

Sanem Arslan Yılmaz

Some of the slides are from:
CMPE150 – Boğaziçi University
Deitel & Associates

Agenda

- Characters
- Strings
  - Definitions & Syntax
  - Initialization
  - Index and bounds
- Character-Handling Library
- String Functions
- String-Conversion Functions
- Standard Input/Output Library Functions
- Search Functions of the String-Handling Library
- Examples

Characters

- Building blocks of source programs.
  - Every program is composed of a sequence of meaningfully grouped characters.

- Character constant
  - An **int** value represented as a character in single quotes
  - **'z'** represents the integer value of **z**

What is a String?

- A string is actually a character array.
  - You can use it like a regular array of characters.
  - However, it has also some unique features that make string processing easy.

- A string may include letters, digits and various special characters such as +, -, *, / and $.

What You Should Keep in Mind

1. Anything written in double quotes (" ") is a string.
2. The end of a string is always marked with the invisible null character **'\0'**.
   - We say "a string is always null-terminated."
   - All string functions depend on this null character. If you ignore the null character, everything will fail.
   - You have to take into account that null character also consumes one byte.
3. Strings should be treated gently. Failure to abide with the rules results in disaster.

Strings

- So, the idea is simple: Obey the following simple rules:
  - When you define the string variable, don't forget to add one byte for the null character.
  - All string functions depend on the null character so don't mess around with the null character.
    - Anything after the null character will be ignored.
    - Anything up to the null character will be considered.

Initializing Strings

- Instead of initializing the elements of a string one-by-one, you can initialize it using a string.
  - A *character array* or a *variable of type char ** can be initialized with a string in a definition.

- Example:

```
char str[] = "JAVA&C";
char str[] = {'J','A','V','A','&','C', '\0'};
char *str = "JAVA&C";
```

Inputting Strings

- A string can be stored in an array using scanf.
- For example, the following statement stores a string in character array `word[20]`:
  - `scanf("%s", word);`
  - The string entered by the user is stored in **word[].**
  - Do not need **&** (because a string is a pointer)
  - Remember to leave room in the array for **'\0'**
- Use `scanf("%19s", word);`
  - `scanf` reads a *maximum* of 19 characters
  - saves the last character for the string's terminating null character
  - prevents `scanf` from writing characters into memory beyond the end of **word**.

Character-Handling Library

- The character-handling library (`<ctype.h>`)
  - includes several functions that perform useful tests and manipulations of character data.

- Each function receives an unsigned char (represented as an int) or EOF as an argument.

- The following slide contains a table of all the functions in **<ctype.h>**

Character-Handling Library (cont.)

Prototype	Function description
`int isblank(int c);`	Returns a true value if `c` is a *blank character* that separates words in a line of text and 0 (false) otherwise. [*Note:* This function is not available in Microsoft Visual C++.]
`int isdigit(int c);`	Returns a true value if `c` is a *digit* and 0 (false) otherwise.
`int isalpha(int c);`	Returns a true value if `c` is a *letter* and 0 (false) otherwise.
`int isalnum(int c);`	Returns a true value if `c` is a *digit* or a *letter* and 0 (false) otherwise.
`int isxdigit(int c);`	Returns a true value if `c` is a *hexadecimal digit character* and 0 (false) otherwise. (See Appendix C for a detailed explanation of binary numbers, octal numbers, decimal numbers and hexadecimal numbers.)
`int islower(int c);`	Returns a true value if `c` is a *lowercase letter* and 0 (false) otherwise.
`int isupper(int c);`	Returns a true value if `c` is an *uppercase letter* and 0 (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.

Character-Handling Library (cont.)

Prototype	Function description
`int isspace(int c);`	Returns a true value if `c` is a *whitespace character*—newline (`'\n'`), space (`' '`), form feed (`'\f'`), carriage return (`'\r'`), horizontal tab (`'\t'`) or vertical tab (`'\v'`)—and 0 (false) otherwise.
`int iscntrl(int c);`	Returns a true value if `c` is a *control character*—horizontal tab (`'\t'`), vertical tab (`'\v'`), form feed (`'\f'`), alert (`'\a'`), backspace (`'\b'`), carriage return (`'\r'`), newline (`'\n'`) and others—and 0 (false) otherwise.
`int ispunct(int c);`	Returns a true value if `c` is a *printing character other than a space, a digit, or a letter*—such as `$`, `#`, `(`, `)`, `[`, `]`, `{`, `}`, `;`, `:` or `%`—and returns 0 otherwise.
`int isprint(int c);`	Returns a true value if `c` is a *printing character* (i.e., a character that's visible on the screen) *including a space* and returns 0 (false) otherwise.
`int isgraph(int c);`	Returns a true value if `c` is a *printing character other than a space* and returns 0 (false) otherwise.

String Functions

- There are multiple string functions that help programming. Learn what type of arguments are required and what they return.

- Anything in double quotes is a string.
  - You may access a string in double quotes.

- You can find several string functions in `string.h`.
  - strlen(), strcpy(), strcat(), strcmp()

String Lenght: strlen()

- `int strlen(char *st)`
  - Returns the length of its string parameter (excluding null character).

- Example: Convert lowercase to uppercase

```
/* Convert lowercase to uppercase */
  for (j=0; j<strlen(st); j++)
     if (('a'<=st[j]) && (st[j]<='z'))
           st[j] += 'A'-'a';
```

String Copy: strcpy()

- If you want to copy the contents of a string variable to another, simple assignment does not work !
  - Example:

```
char st1[5]="abcd", st2[5]="xyz";
st1=st2;
```

    is wrong.

    You have to copy the characters one-by-one.

    There is a specific function that does this.

- `char *strcpy(char *dest, const char *source)`
  - Copies all characters in `source` into `dest`.
  - Of course terminates `dest` with null char.
  - Returns starting address of `dest`.

Example - strcpy()

```
char st1[6]="abdef", st2[5]="xyz";
strcpy(st1,st2);
st1[2]='M';
st2[3]='N';
printf("<st1:%s>\n",st1);
printf("<st2:%s>\n",st2);
```

What is the output?

```
<st1:xyM>
<st2:xyzN>
```

String Concatenation: strcat()

- If you want to attach two strings, use strcat().

- `char *strcat(char *dest, char *source)`
  - Attaches `source` to the tail of `dest`.
  - Chars in `dest` are not lost.
  - Returns starting address of `dest`.

Example - strcat()

- Write a function that reads a name from the input, prepends it with "Hello ", and updates its parameter to contain this greeting string. (You may assume the caller passes a parameter that is large enough.)

```
void greet(char g_st[])
{    char name[20];
     scanf("%s", name);
     strcpy(g_st,"Hello ");
     strcat(g_st, name);
}
```

String Compare: strcmp()

- You may also check if the lexicographical ordering of two strings.
- `int strcmp(const char *st1, const char *st2)`
  - Returns <0 if `st1` comes before `st2`.
  - Returns 0 if `st1` is identical to `st2`.
  - Returns >0 if `st1` comes after `st2`.

Variants of String Functions

- For string functions, you must ensure that the character array is large enough to store the string and its terminating null character.
  - more data is stored in an array than its declared size allows
  - a very dangerous condition

- A solution is the use of safer functions: strncpy(), strncat(), strncmp()
  - `strncpy(dest,src,n)`
    - Copy at most `n` characters of `src` to `dest.`
  - `strncat(dest,src,n)`
    - Concatenate at most `n` characters of `src` to `dest.`
  - `strncmp(dest,src,n)`
    - Compare at most `n` characters of `dest.`

String-Conversion Functions

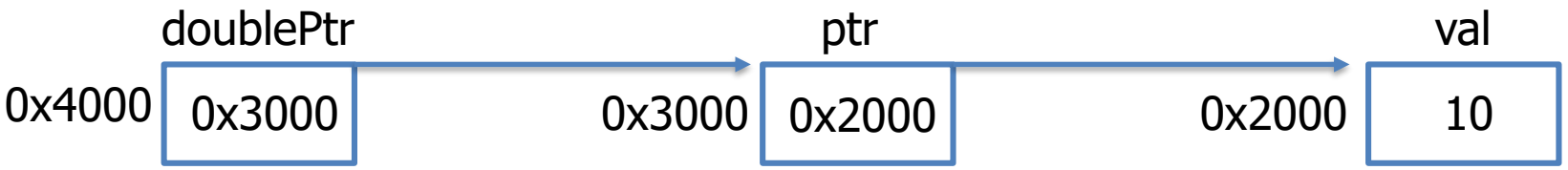
- Conversion functions
  - In **<stdlib.h>** (general utilities library)
  - Convert strings of digits to integer and floatingpoint values

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**.

Double Pointer (Pointer to Pointer)

- ** → the pointer to store the address of another pointer
- Example:

```
int **doublePtr;
int *ptr;
int var = 10;
ptr = &var;
doublePtr = &ptr;
printf("Value of var = %d\n", var );  // prints 10
printf("Value of var using single pointer = %d\n", *ptr); // prints 10
printf("Value of var using double pointer = %d\n", **doublePtr); // prints 10
```



Example: fig08_06.c

```c
// Fig. 8.6: fig08_06.c
// Using function strtod
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
   const char *string = "51.2% are admitted"; // initialize string
   char *stringPtr; // create char pointer

   double d = strtod(string, &stringPtr);

   printf("The string \"%s\" is converted to the\n", string);
   printf("double value %.2f and the string \"%s\"\n", d, stringPtr);
}
```

```
The string "51.2% are admitted" is converted to the
double value 51.20 and the string "% are admitted"
```

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 *fgets(char *s, int n, FILE *stream);`	Inputs characters from the specified stream into the array `s` until a *newline* or *end-of-file* character is encountered, or until `n - 1` bytes are read. In this chapter, we specify the stream as `stdin`—the *standard input stream*, which is typically used to read characters from the keyboard. A *terminating null character* is appended to the array. Returns the string that was read into `s`. If a newline is encountered, it's included in the string stored in `s`.
`int putchar(int c);`	Prints the character stored in `c` and returns it as an integer.
`int puts(const char *s);`	Prints the string `s` followed by a *newline* character. Returns a non-zero integer if successful, or `EOF` if an error occurs.

Standard Input/Output Library Functions (cont.)

Function prototype	Function description
`int sprintf(char *s, const char *format, ...);`	Equivalent to `printf`, except the output is stored in the array `s` instead of printed on the screen. Returns the number of characters written to `s`, or `EOF` if an error occurs. [*Note:* We mention the more secure related functions in the Secure C Programming section of this chapter.]
`int sscanf(char *s, const char *format, ...);`	Equivalent to `scanf`, except the input is read from the array `s` rather than from the keyboard. Returns the number of items successfully read by the function, or `EOF` if an error occurs. [*Note:* We mention the more secure related functions in the Secure C Programming section of this chapter.]

Example: fig08_10.c

```
1  // Fig. 8.10: fig08_10.c
2  // Using functions fgets and putchar
3  #include <stdio.h>
4  #define SIZE 80
5
6  void reverse(const char * const sPtr); // prototype
7
8  int main(void)
9  {
10    char sentence[SIZE]; // create char array
11
12    puts("Enter a line of text:");
13
14    // use fgets to read line of text
15    fgets(sentence, SIZE, stdin);
16
17    printf("\n%s", "The line printed backward is:");
18    reverse(sentence);
19 }
20
```

Example: fig08_10.c (cont.)

```
21  // recursively outputs characters in string in reverse order
22  void reverse(const char * const sPtr)
23  {
24     // if end of the string
25     if ('\0' == sPtr[0]) { // base case
26        return;
27     }
28     else { // if not end of the string
29        reverse(&sPtr[1]); // recursion step
30        putchar(sPtr[0]); // use putchar to display character
31     }
32  }
```

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

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

Example: fig08_12.c

```
1  // Fig. 8.12: fig08_12.c
2  // Using function sprintf
3  #include <stdio.h>
4  #define SIZE 80
5
6  int main(void)
7  {
8     int x; // x value to be input
9     double y; // y value to be input
10
11    puts("Enter an integer and a double:");
12    scanf("%d%lf", &x, &y);
13
14    char s[SIZE]; // create char array
15    sprintf(s, "integer:%6d\ndouble:%7.2f", x, y);
16
17    printf("%s\n%s\n", "The formatted output stored in array s is:", s);
18
19 }
```

```
Enter an integer and a double:
298 87.375
The formatted output stored in array s is:
integer:   298
double:  87.38
```

Search Functions of the String-Handling Library

Function prototypes and descriptions

```
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.

Search Functions of the String-Handling Library (cont.)

Function prototypes and descriptions

```
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.

Example: fig08_26.c

```c
// Fig. 8.26: fig08_26.c
// Using function strtok
#include <stdio.h>
#include <string.h>

int main(void)
{
   // initialize array string
   char string[] = "This is a sentence with 7 tokens";

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

   char *tokenPtr = strtok(string, " "); // begin tokenizing sentence

   // continue tokenizing sentence until tokenPtr becomes NULL
   while (tokenPtr != NULL) {
      printf("%s\n", tokenPtr);
      tokenPtr = strtok(NULL, " "); // get next token
   }
}
```

Example: fig08_26.c (cont.)

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

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

Example - 1

- Write a program that reads a string from the user.
- Your program will detect whether the input string is a palindrome or not.
- A palindrome is a character sequence which is symmetric eg:
  - abcddcba
  - aaaappaaaa
  - xyzyx
  - ac bb ca

Example - 2

- Write a program which compresses the entered character sequence with the Run Length Encoding (RLE) algorithm.

- RLE algorithm: It replaces sequences of the same data values (eg: characters, integers etc) by a count number and a single value.

- **EXAMPLE RUN:**
  INPUT:
  aaaXXyyyyyZ+++bb++++++77
  OUTPUT:
  3a2X4y1Z3+2b5+27