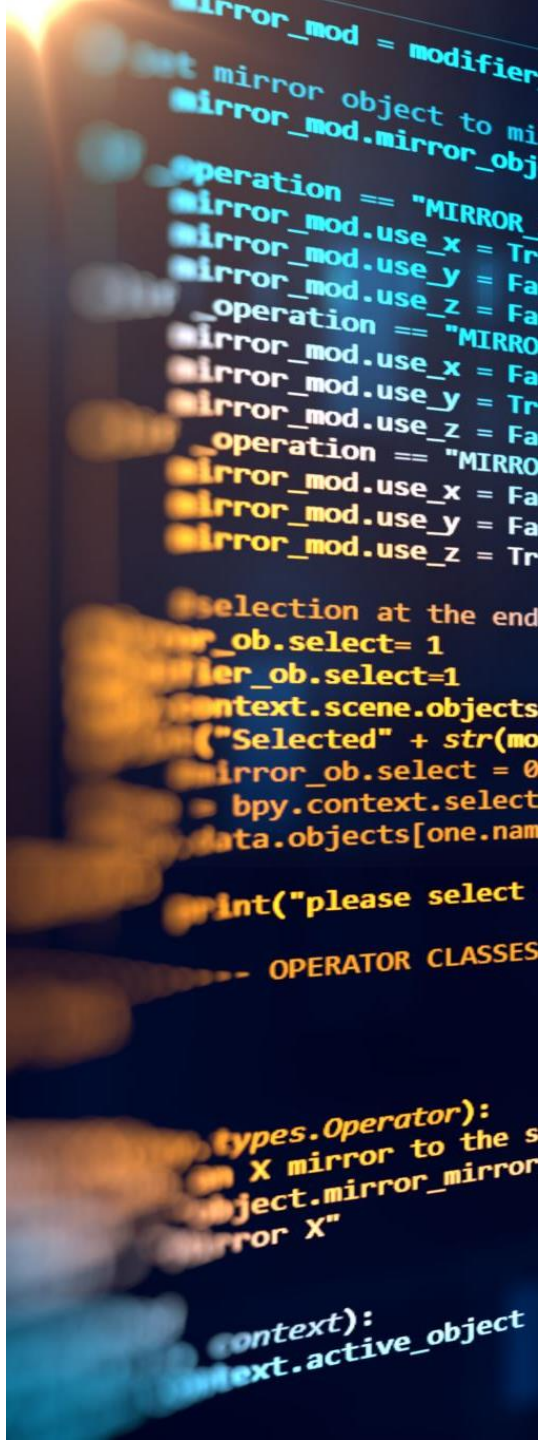
CS 1037
Fundamentals of Computer Science II

C Programming Features

Ahmed Ibrahim



C Advances (cont.)

Strings

- Almost all messages or text information we see on computer screens are represented internally in a computer as strings.
- **What is a string?** a **string** is data consisting of a sequence of characters, e.g., “hello”
- In C program language, a **string** is a sequence of non-null characters followed by a null character (‘\0’) stored in a char array.

```c
#include <stdio.h>
int main(){
  char str[10];
  str[0] = 'H';
  str[1] = 'e';
  str[2] = 'l';
  str[3] = 'l';
  str[4] = 'o';
  str[5] = '\0';
  printf("%s\n", str);  // output: Hello

  int i;
  for (i=0; i<10; i++) {
     if (str[i] != '\0')
       printf("index: %d, char: %c, code:%d, address: %lu\n",
    i, str[i], str[i], &str[i]);
    else
    break;
  }
  return 0;
}
```

str — array

0	1	2	3	4	5	6	7	8	9
char	char	char	char	char	char	char	char	char	char
'H'	'e'	'l'	'l'	'o'	'\0'	?	?	?	?

Strings

- The **null character** is the character with ASCII code `0`, represented by (‘`\0`’).
- The **length** of a string is defined as the number of non-null characters.
- The index of a character in a string is the position number from the beginning of the string.
- For example, the string “Hello” is 11 long, and the index of the letter H is `0`.

```c
#include <stdio.h>

int main() {
    char str[10] = "Hello";

    // output: Hello
    printf("%s\n", str);

    int i;
    for (i = 0; i < 10; i++) {
        // Correct null character check
        if (str[i] == '\0')
            break;
        else
            printf("index: %d, char: %c, code: %d, address: %lu\n",
            i, str[i], str[i], &str[i]);
    }
    return 0;
}
```

str	array									
	0	1	2	3	4	5	6	7	8	9
	char	char	char	char	char	char	char	char	char	char
	'H'	'e'	'l'	'l'	'o'	'\0'	'\0'	'\0'	'\0'	'\0'

```
Hello
index: 0, char: H, code: 72, address: 68702702542
index: 1, char: e, code: 101, address: 68702702543
index: 2, char: l, code: 108, address: 68702702544
index: 3, char: l, code: 108, address: 68702702545
index: 4, char: o, code: 111, address: 68702702546
```

Or `char str[10] = {'H','e','l','l','o','\0'};`

```c
#include <stdio.h>

int main() {
    char str[10];
    str[0] = 'H';
    str[1] = 'e';
    str[2] = 'l';
    str[3] = 'l';
    str[4] = 'o';
    str[5] = '\0';

    // output: Hello
    printf("%s\n", str);

    int i;
    for (i = 0; i < 10; i++) {
      // Correct null character check
        if (str[i] == '\0')
          break;
        else
          printf("index: %d, char: %c, code: %d, address: %lu\n",
            i, str[i], str[i], &str[i]);
    }
    return 0;
}
```

str	array									
	0	1	2	3	4	5	6	7	8	9
	char	char	char	char	char	char	char	char	char	char
	'H'	'e'	'l'	'l'	'o'	'\0'	?	?	?	?

double quotes

```c
#include <stdio.h>

int main() {
    char str[10] = "Hello";

    // output: Hello
    printf("%s\n", str);

    int i;
    for (i = 0; i < 10; i++) {
      // Correct null character check
        if (str[i] == '\0')
          break;
        else
          printf("index: %d, char: %c, code: %d, address: %lu\n",
            i, str[i], str[i], &str[i]);
    }
    return 0;
}
```

str	array									
	0	1	2	3	4	5	6	7	8	9
	char	char	char	char	char	char	char	char	char	char
	'H'	'e'	'l'	'l'	'o'	'\0'	'\0'	'\0'	'\0'	'\0'

Char array length

- When the array length is **not specified**, the compiler sets the array length to the string length plus one.
- Example: For the statement char str[] = "Hello"; The compiler allocates 6 bytes for the char array str.
- Characters 'H', 'e', 'l', 'l', 'o' are placed into str[i] at indices i = 0, 1, 2, 3, 4.
- The null character '\0' is placed at str[5].
- This is equivalent to the statement: char str[] = {'H', 'e', 'l', 'l', 'o', '\0'};.

Array of strings

- Assume that we want to store and process a collection of names. It is better to store each name as an individual string so that it is efficient to sort, search, and perform other operations on the names.
- An array of strings is a sequence of strings stored in a 2D char array.
- An array of strings can be declared and initialized with a list of string expressions.

```c
// command_line_argument.c
#include<stdio.h>
int main()
{
  char names[3][6] = {"Jorge", "Dan", "Brian"};
  printf("%s\n", names[0]);    // output: Jorge
  printf("%s\n", names[1]);    // output: Dan
  printf("%s\n", names[2]);    // output: Brian
  printf("%c", *(names[1]+2));  // output: n
    return 0;
}
```

Output:

```
Jorge
Dan
Brian
n
```

Memory:

	array					
names	0,0 char 'J'	0,1 char 'o'	0,2 char 'r'	0,3 char 'g'	0,4 char 'e'	0,5 char '\0'
	1,0 char 'D'	1,1 char 'a'	1,2 char 'n'	1,3 char '\0'	1,4 char '\0'	1,5 char '\0'
	2,0 char 'B'	2,1 char 'r'	2,2 char 'i'	2,3 char 'a'	2,4 char 'n'	2,5 char '\0'

String operations by pointers

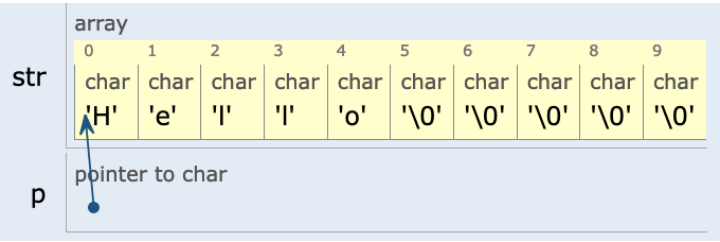
- Since a string is stored in a char array, all array operations and pointer operations on arrays apply to string operations.
- The following code fragment shows the usage of pointers in string operations.

```c
#include <stdio.h>
int main(){
 char str[10] = "Hello";
 char *p;
 p = &str[0];

 printf("%c\n", *p);
 printf("%s\n", p);
 printf("%c\n", *(p+1));
 printf("%s\n", p+1);
  return 0;
}
```

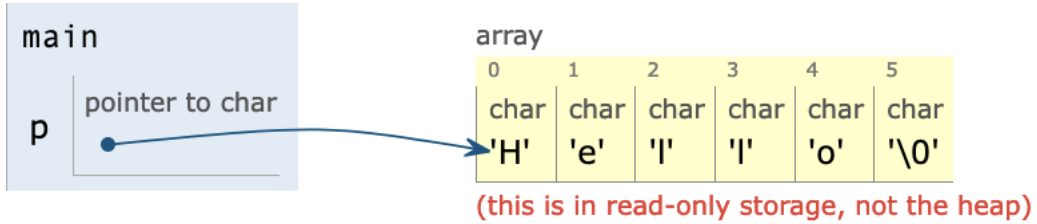
Output:

```
H
Hello
e
ello
```



String pointers

- In C, a string can be created and referenced by a char pointer, allowing access to the string through the pointer.
- Example:



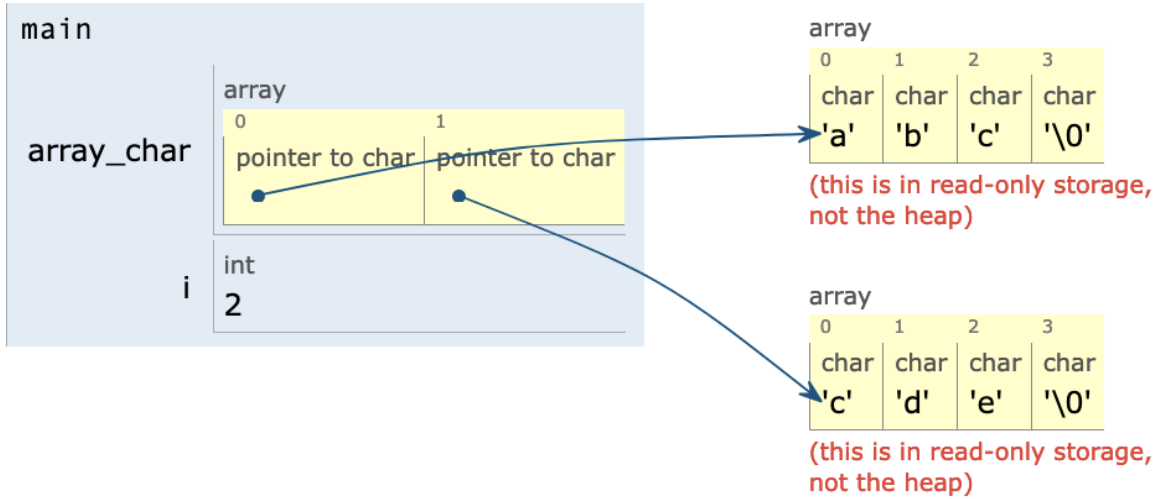

```c
#include <stdio.h>
int main(){
    char *p = "Hello";
    printf("%s\n", p);        // output: Hello
    printf("%c\n", *(p+2)); // output: l
    *p = 'h';                 // this statement is not allowed.
  return 0;
}
```

- string literals (such as "Hello" in your example) are stored in a read-only section of memory because they are considered constant by the compiler.

Array of string pointers

- An array of string pointers is an array of char pointers. Each element of the array is a pointer to a char array.
- Example:

```c
#include <stdio.h>
int main(){
    char *array_char[] = {"abc", "cde"};
    int i;
    for (i=0; i<2; i++)
  printf("%s\n", array_char[i]);
  return 0;
}
```



Application of string pointers

- One application of the array of string pointers is the command line arguments storage and access.
- In C programming, input data can be passed to the program by the command line. The syntax is as follows.

```
int main(int argc, char *argv[]) { /* ... */ }
```

- argc, an integer type, serves as a counter for the number of command-line arguments, including the program's name.
- On the other hand, argv[] is an <u>array of character pointers</u> that holds all the arguments, with argv[0] being the program's name and argv[1] to argv[argc-1] representing the command line arguments.

```c
// command_line_argument.c
#include<stdio.h>
int main(int argc, char* argv[])
{
  int i;
  if(argc > 0)
  {
    for(i=0; i<argc; i++)
      printf("argv[%d]: %s\n", i, argv[i]);
    }
    return 0;
}
```

Output:

Print output (drag lower right corner to resize)

```
argv[0]: /tmp/opt-cpp-backend/usercode.exe
```

Stack Heap

main

argc	int 1
argv	pointer to char* 💀
i	int 1

String Operations

String Operations

String operations are common in applications.

The following are basic string operations.

- Read a string from stdin
- Write a string to stdout
- Get the length of a string
- Copy strings
- Compare two strings

Reading a string from stdin

- Here stdin represents the standard input device, default to be the keyboard.
- C's standard I/O library stdio provides three functions to get input from the keyboard.
  - scanf() – get formatted data, e.g., scanf("%s", str); prompts the user to type a **string** and hit the enter key to terminate the input.
  - gets() – get a **string**, e.g., gets(str); prompts the user to type a **string** and hit the enter key to terminate the input.
  - getchar() – get and return a character, e.g., str[0] = getchar();

```c
#include <stdio.h>

int main() {
    char str1[50], str2[50], ch;

    printf("Enter a string (scanf): ");
    scanf("%s", str1);
    printf("scanf: %s\n", str1);

    while ((getchar()) != '\n'); // Clear bu

    printf("Enter a string (gets): ");
    gets(str2);  // Note: gets() is deprecat
    printf("gets: %s\n", str2);

    printf("Enter a character (getchar): ");
    ch = getchar();
    printf("getchar: %c\n", ch);

    return 0;
}
```

Writing a string to stdout

- stdout represents the standard output, i.e., screen. The basic operation is to print a single character on the screen. stdio function putchar(char) prints a **single** character on screen.

```
void display_string(char s[]) {
  int i;
  for (i=0; s[i] != '\0'; i++ ) {
      // put character s[i] to screen
    putchar(s[i]);
  }
}
```

- The algorithm for printing a string to stdout is to traverse the string, write each character on stdout, and stop when null is encountered.
- stdio library functions printf() and puts() also print strings on screen.

```
printf("Using printf: %s\n", str); // Needs explicit newline
puts("Using puts: Hello, World!"); // Automatically adds newline
```

Printf vs. Puts

Feature	printf	puts
Usage	Prints formatted output	Prints a string with a newline
Newline	Does not add automatically (\n needed)	Adds a newline automatically
Formatting	Supports format specifiers	No formatting support
Return Value	Number of characters printed	1 on success, -1 EOF on error

Getting the length of a string

- To get the length, increase a counter by 1 for each non-null character. The following function implements the algorithm.

```
int get_length(char *s) {
  if (s == NULL) return -1;
  int counter = 0;
  while (*s) {
    counter++;
    s++;
  }
  return counter;
}
```

Copying string

- The algorithm is to traverse the source string, copy the character to the destination array, and add null to the end.
- The following function shows a straight implementation of the algorithm.
- It assumes that the destination char array has enough space to hold the source string.

```c
#include <stdio.h>

int main() {
    char source[] = "Hello, World!";
    // Ensure this array is large enough
    // to hold the source string
    char destination[50];
    // Copying the string using a loop
    int i = 0;
    while (source[i] != '\0') {
        destination[i] = source[i];
        i++;
    }

    // Don't forget to add
    // the null character at the end
    destination[i] = '\0';

    printf("Source string: %s\n", source);
    printf("Copied string: %s\n", destination);

    return 0;
}
```

Comparing two strings

- Given two strings, s1 and s2.
- Comparing s1 and s2 returns results according to the following:

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

int main() {
    char str1[] = "Hello";
    char str2[] = "hello";

    // Using strcmp() to compare strings
    int result = strcmp(str1, str2);

    if (result == 0)
        printf("The strings are equal.\n");
    else if (result > 0)
        printf("str1 is greater than str2.\n");
    else
        printf("str1 is less than str2.\n");

    return 0;
}
```

Stop and think

- How do you append a string to the end of another string (concatenate)?
- How do you convert characters of a string into upper case?
- How do you reverse a string?
- How do you convert a string number

like “264” to integer 264?

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

int main() {
    char str[] = "264";
    int num = atoi(str);  // Convert string to integer

    printf("The integer value is: %d\n", num);

    return 0;
}
```

```c
#include <stdio.h>

int main() {
    char str[] = "hello, world!";
    int i = 0;

    while (str[i] != '\0') {
        // Check if the character is lowercase
        if (str[i] >= 'a' && str[i] <= 'z') {
          ; // Convert to uppercase by subtracting 32
            str[i] = str[i] - 32
        }
        i++;
    }

    printf("Uppercase string: %s\n", str);

    return 0;
}
```

Question!

Given:

```
char str[] = "abc";
```

which of the following is not correct?

A) sizeof(s) is 4 bytes

B) sizeof(s) is 3 bytes

C) the length of string str is 3

D) str[3] holds the null character '\0'

Question!

Given the following code snippet:

```
char str1[] = "hello";
char *str2 = "hello";
str1[0] = 'H';
str2[0] = 'H';
```

What will be the outcome of this code?

A) Both str1 and str2 will be changed to "Hello".

B) Only str1 will be changed to "Hello", and str2 will cause a runtime error.

C) Both str1 and str2 will cause a runtime error.

D) Only str2 will change to "Hello", while str1 remains "hello".

String Library

string.h

Since strings and string operations are commonly used in applications, C provides a string library for commonly used string operations.

The header file of the library is string.h, use preprocessing directive #include <string.h> to include the header file.

String Library

- String length function

```
char name[20] = "data structures";
printf("%d", strlen(name));
```

- String copy function

```
char name[20];
strcpy(name, "data structures");
printf("%s", name);
```

- Note that C does not allow array assignment after declaration.

- Concatenate two strings

```
char s1[20] = "hello,";
char s2[20] = "world";
strcat(s1, s2);
```

- Compare two strings

```
char s1[20] = "hello ";
char s2[20] = "world";
printf("%d",  strcmp(s1, s2));
```


thank you

References

- Data Structures Using C, second edition, by Reema Thareja, Oxford University Press, 2014.