

CSE216

Foundations of Computer Science

Instructor: Zhoulai Fu

State University of New York, Korea

C crash course (cont.)

Errata:

Indeed, C99 has a bool type

- It is called `_Bool`
- Aliased to `bool` in `stdbool.h`
- But no `bool` before C99. E.g. No `bool` in C89 (ANSI C).

In the following

- **C Pointers**
- **Lab questions**

FANG Interview Question 1: **const char* vs. char* const**

- What does **const char* s = "hello";** do ?
- **const char*** is a pointer to a "constant character". This means you're not allowed to change the character that the pointer is referring to

FANG Interview Question 1: **const char* vs. char* const**

- What does **char* const str = "hello";** do ?
- **char* const** is a “constant pointer” to a character. This means you're not allowed to change the pointer

FANG Interview Question 2:

What is void*?

- generic pointer type
- raw address in memory
- can store the address of any object
- can be type-casted to any type of pointer
- Cannot be directly dereferenced. To access the object, you must first cast void * to a correct pointer type.

```
1  #include <stdio.h>
2
3  void printNumber(void *ptr, char type) {
4      if (type == 'i') { // If the type is integer
5          int *int_ptr = (int *)ptr;
6          printf("The number is %d\n", *int_ptr);
7      } else if (type == 'f') { // If the type is float
8          float *float_ptr = (float *)ptr;
9          printf("The number is %f\n", *float_ptr);
10     }
11 }
12
13 int main() {
14     int i = 5;
15     float f = 9.5;
16
17     // Passing integer pointer
18     printNumber(&i, 'i');
19
20     // Passing float pointer
21     printNumber(&f, 'f');
22
23     return 0;
24 }
```


Address of (&)

- To get the address of a variable, we use the ampersand (&) operator,

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

```
int main() {  
    int var = 5;  
    printf("Address of var: %p\n", (void*)&var);  
    return 0;  
}
```

Pointers (*)

- A pointer is a variable whose value is an address
- To define a pointer, use **type *var-name;**

```
int    *ip;    /* pointer to an integer */  
double *dp;    /* pointer to a double */  
float  *fp;    /* pointer to a float */  
char   *ch;    /* pointer to a character */
```

Question

- If the 1st printf gives 0x7fff5c7ea38c, what will be the results of the 2nd and 3rd printf?

```
#include <stdio.h>
int main () {
    int var = 20;
    int *ip = &var;

    printf("Address of var variable: %p\n", (void*) &var );

    printf("Address stored in ip variable: %p\n", (void *) ip );

    printf("Value of *ip variable: %d\n", *ip );
    return 0;
}
```

Question

- Any difference between `int *ptr` and `int* ptr`?
- The default usage is `int *ptr`; although many prefer the other
- What is the type of “b” in **`int* a, b`**?

NULL Pointer

- The NULL pointer always points to no objects
- `int *ptr = NULL;`
- cannot be dereferenced

Non-sense from an online tutorial

The **NULL** pointer is a constant with a value of zero defined in several standard libraries. Consider the following program:

```
#include <stdio.h>

int main ()
{
    int *ptr = NULL;

    printf("The value of ptr is : %x\n", &ptr );

    return 0;
}
```

WRONG

When the above code is compiled and executed, it produces the following result:

The value of ptr is 0

WRONG

Question: What will happen if we run this

```
#include <stdio.h>
int main () {

    int *ptr = NULL;

    printf("The value of ptr is : %p\n", (void *) &ptr );

    printf("The value of ptr is : %p\n", (void *) ptr );

    printf("The value of ptr is : %x\n", *ptr );

    return 0;
}
```

Try: jdoodle.com/ia/IN3

Check if a pointer is Null

```
if(ptr)      /* succeeds if p is not null */  
if(!ptr)     /* succeeds if p is null */
```


FANG Interview Question 3:

Pointer arithmetics

- Assume ptr is an 32-bit integer pointer which points to the address 0x1009.
- $\text{ptr}++ = ?$
- Assume ptr is a char pointer which points to the address 1009.
- $\text{ptr}++ = ?$

FANG Interview Question 4:

Which one will crash, and why?

```
1 #include <stdio.h>
2
3 const int MAX = 3;
4 int main () {
5
6     int var[] = {10, 100, 200};
7
8     double b[5] = {1000.0, 2.0, 3.4, 17.0, 50.0};
9     double *ptr=b;
10
11     for (int i = 0; i < 5; i++) {
12         printf("Value of balance[%d] = %f\n", i, *ptr );
13         /* move to the next location */
14         ptr++;
15     }
16     return 0;
17 }
```

```
1 #include <stdio.h>
2
3 const int MAX = 3;
4 int main () {
5
6     int var[] = {10, 100, 200};
7
8     double b[5] = {1000.0, 2.0, 3.4, 17.0, 50.0};
9
10     for (int i = 0; i < 5; i++) {
11         printf("Value of b[%d] = %f\n", i, *b );
12         /* move to the next location */
13         b++;
14     }
15     return 0;
16 }
```

Arrays

- Arrays are constant pointers
- `int x[5];`
- `x++` would not work
- Use `int * ptr = x; ptr++;`

C Exercises

1

- What will be the output of the following C code?

```
int main() {  
    int var = 20;  
    int *ptr;  
    ptr = &var;  
    printf("%d", *ptr);  
    return 0;  
}
```

2

- What will be the output of the following C code?

```
int x = 10;
```

```
int *p = &x;
```

```
*p = 20;
```

```
printf("%d", x);
```

3

- What will be the output of the following C code?

```
int main() {  
  
    int arr[] = {10, 20, 30, 40, 50, 60};  
  
    int *ptr = arr;  
  
    printf("%d ", *(ptr++));  
  
    printf("%d", *ptr);  
  
    return 0;  
  
}
```

4

- What will be the output of the following C code?

```
int x = 10;
```

```
int *p = &x;
```

```
*p = 20;
```

```
printf("%d", x);
```


Lab exercise 1:

Implementing a Caesar Cipher in C

- In this lab exercise, you will be implementing a simple Caesar cipher in the C programming language. A Caesar cipher is a type of substitution cipher where each character in the plaintext is 'shifted' a certain number of places down the alphabet. For example, with a shift of 3, A would be replaced by D, B would become E, and so on.

Problem Statement

```
int main() {  
    char str[] = "KENNEDY";  
    caesarCipher(str);  
    printf("%s\n", str); // Should print "NHQQHGB"  
    return 0;  
}
```

- Write a C function **void caesarCipher(char* str)** that performs a Caesar cipher on an input string. The string will consist of capital letters only, and the cipher should shift each letter 3 places to the right in the alphabet, wrapping around to the beginning of the alphabet if necessary.
- For example, the input string "KENNEDY" should produce the output "NHQQHGB".

FYI

- **Character arrays in C:** In C, strings are typically represented as arrays of characters. For example, the string "HELLO" can be declared as **char str[] = "HELLO";**. Note that all strings in C are null-terminated, which means they end with a special character '\0'.
- **Character pointers in C (char*):** A character pointer in C can also be used to represent a string. It can point to the first character of a string, and the string is assumed to continue until a null character is encountered. For example, **char* str = "HELLO";**.
- **String manipulation in C:** C provides several functions for manipulating strings, such as **strcpy** for copying strings and **strlen** for finding the length of a string. However, in this exercise, you will be manipulating strings directly.

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

```
void caesarCipher(char* str) {
```

```
    int i = 0;
```

```
    while(str[i] != '\0') {
```

```
        if(str[i] >= 'A' && str[i] <= 'Z') {
```

```
            // Shift character by 3 places. If it goes beyond 'Z', take it back  
to the start
```

```
            str[i] = 'A' + (str[i] - 'A' + 3) % 26;
```

```
        }
```

```
        i++;
```

```
    }
```

```
}
```

```
int main() {
```

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

```
    caesarCipher(str);
```

```
    printf("%s\n", str); // Should print "NHQQHGB"
```

```
    return 0;
```

```
}
```

Lab exercise 2:

Sentence Title Case Verification in C

- Your task is to write a C function that checks whether a sentence is in 'Title Case'. In other words, the function should return true if each word in the sentence starts with a capital letter and continues with lowercase letters. Here are the specific requirements:
 - The function should take a single argument - a string, representing the sentence to check. This string consists only of letters and blank spaces.
 - The function should return a boolean value (in C, typically represented as an int with 0 for false and non-zero for true).
 - The function should return true if and only if each word in the sentence starts with a capital letter and continues with lowercase letters. Otherwise, it should return false.
- Write the function as described above. Test your function with several test sentences to ensure that it works correctly.

FYI

- In C, strings are represented as arrays of characters. You can use array indexing to access individual characters in a string, similar to how you'd access elements in an array. For example, `sentence[0]` would give you the first character in the string `sentence`.
- C provides functions to manipulate and check characters. You might find the following functions from the `ctype.h` library useful:
 - **`isupper(int c)`** checks if the given character is uppercase.
 - **`islower(int c)`** checks if the given character is lowercase.
 - **`isspace(int c)`** checks if the given character is a whitespace character.
- Reminder: A string in C is null-terminated, meaning it ends with the special null character `'\0'`. You can use this fact to iterate through the string.

```

#include <ctype.h>
#include <stdbool.h>
#include <stdio.h>
bool isTitleCase(char* str) {
    int i = 0;
    while (str[i]) {
        // If the character is a space, skip over all spaces and check if next character is uppercase
        if (str[i] == ' ') {
            while (str[i] == ' ') {
                i++;
            }
            if (!str[i] || !isupper(str[i])) {
                return false;
            }
        } else if (i == 0) {
            // The first character of the string should be uppercase
            if (!isupper(str[i])) {
                return false;
            }
        } else if (!islower(str[i])) {
            // All other characters should be lowercase
            return false;
        }
        i++;
    }
    return true;
}

int main(){
    char * s= "Hellon  World";
    char * s2= "Hello world";
    if (isTitleCase(s)) printf ("%s is title case\n", s);
    else printf ("%s is not title case\n", s);
    if (isTitleCase(s2)) printf ("%s is title case\n", s2);
    else printf ("%s is not title case\n", s2);
}

```