CSE216
Foundations of Computer Science

Instructor: Zhoulai Fu

State University of New York, Korea

C crash course

- C Language Overview.
- C Environment Setup
- C Program Structure
- C Basic Syntax
- C Data Types
- C Variables
- C Constants and Literals
- C Storage Classes
- C Operators
- Decision Making in C
- C Loops
- C Functions
- C Scope Rules
- C Arrays
- C Pointers
- C Strings
- C unions

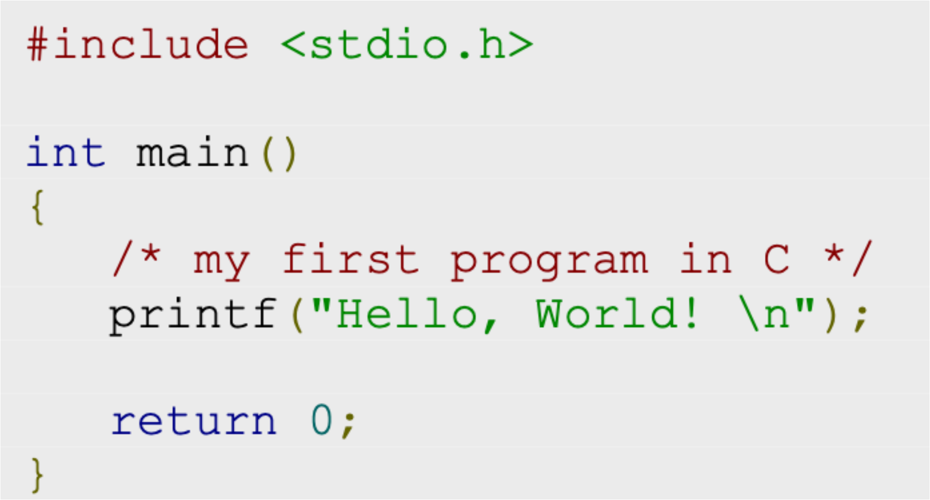
Language Overview

- Dinosaur
- Imperative.
- Close to machine

Environment Setup

- Text editor
- C compiler
- C99 (not ANSI C = C89/C90, not C11, not C17)
- gcc -std=c99 main.c
- https://www.jdoodle.com/compile-c99-online/

Program Structure



```c
#include <stdio.h>

int main()
{
   /* my first program in C */
   printf("Hello, World! \n");

   return 0;
}
```

- Preprocessor Commands
- Functions
- Variables
- Statements & Expressions
- Comments

Basic Syntax

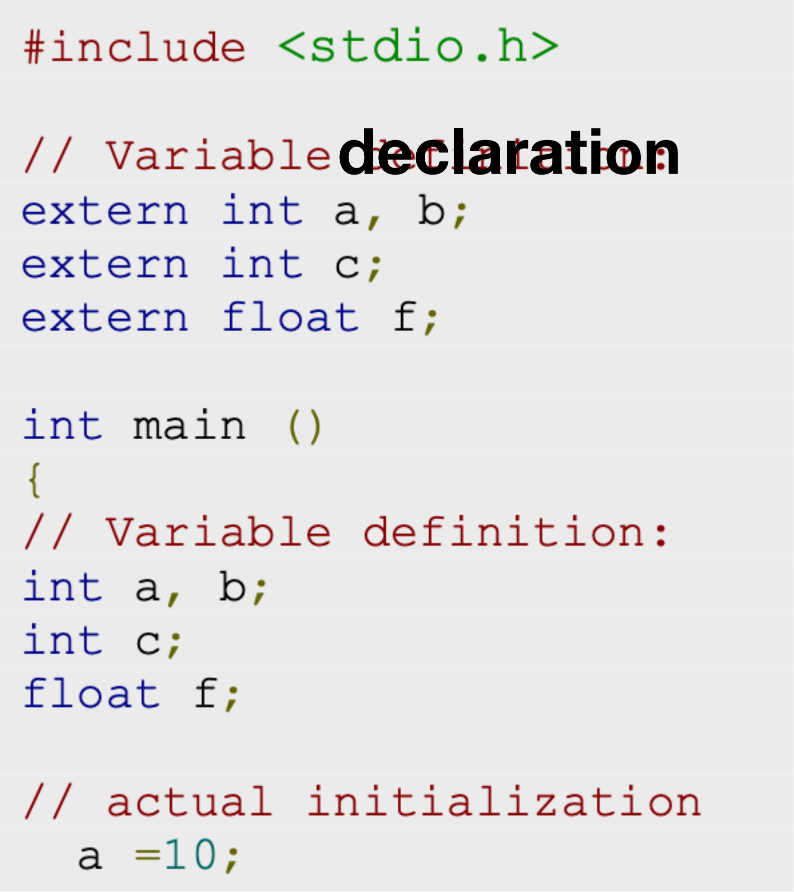
- Program := Statements
- Statement := Tokens
- Token := Keyword | Identifier | Constant | Symbol
- Semicolon ; is statement terminator
- Comments “//…” or “/*…*/” is removed during preprocessing

Data Types

- Basic types: *int, char, float, double, long*… No bool. No string
- Void type: *void exit (int), int rand(void), void *malloc(1024)*
- enum type: *enum mbti {ESTP, INFJ…}*
- *Derived types: Pointer types, Array types, Structure types, Union types and Function types.*

Variables

- Var. declaration
- Var. definition
- Var. initialization



```
#include <stdio.h>

// Variable declaration:
extern int a, b;
extern int c;
extern float f;

int main ()
{
// Variable definition:
int a, b;
int c;
float f;

// actual initialization
  a =10;
```

Question

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

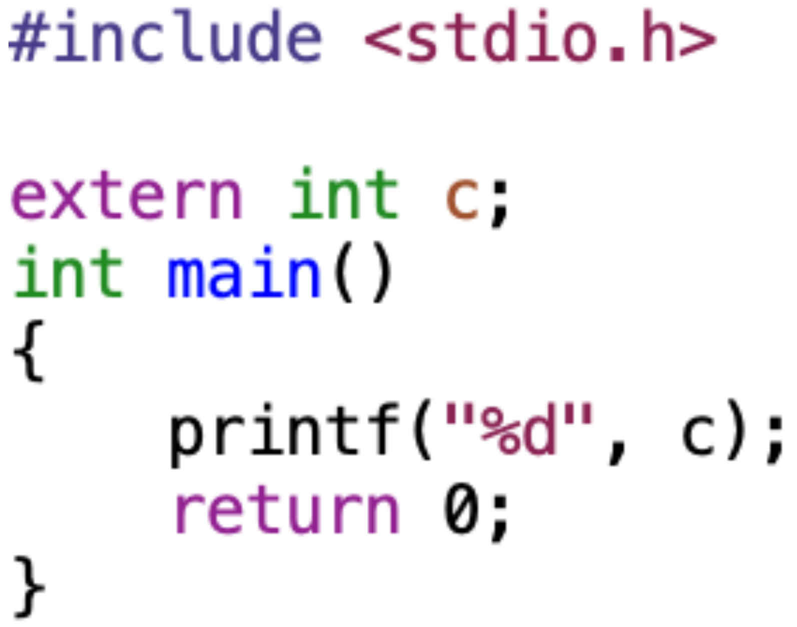
vs.

```
#include “stdio.h”
```

“” looks through current directory, while <> looks through system library folders

Question

- What will happen with “gcc main.c”



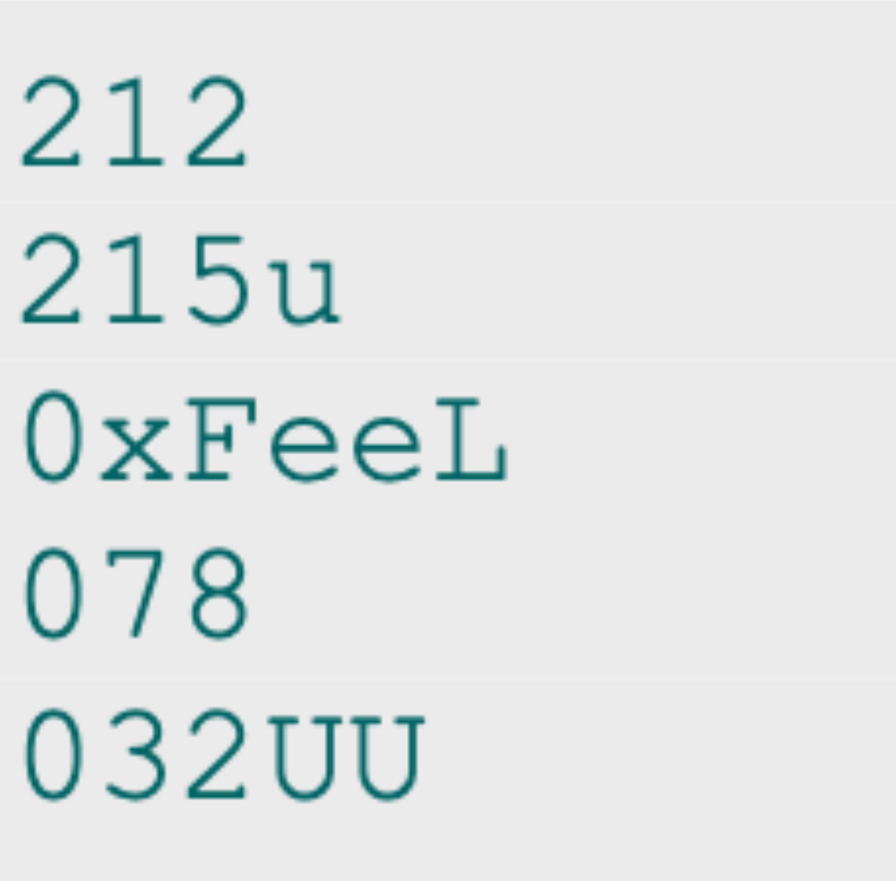
```c
#include <stdio.h>

extern int c;
int main()
{
    printf("%d", c);
    return 0;
}
```

Constants: Integer Literals

- decimal
- octal: 0213
- hexadecimal: 0x4b, 0xA0F
- Unsigned: 30u, 30U
- LongL 42L, 42l
- Suffix is case-insensitive and can be in any order: 30ul — unsigned long

Question

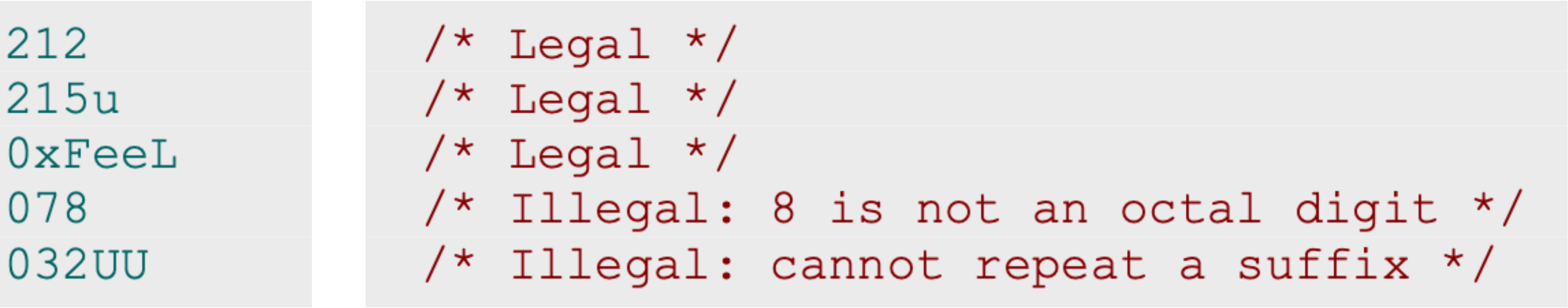
- Which one is illegal?



```
212
215u
0xFeeL
078
032UU
```

Question

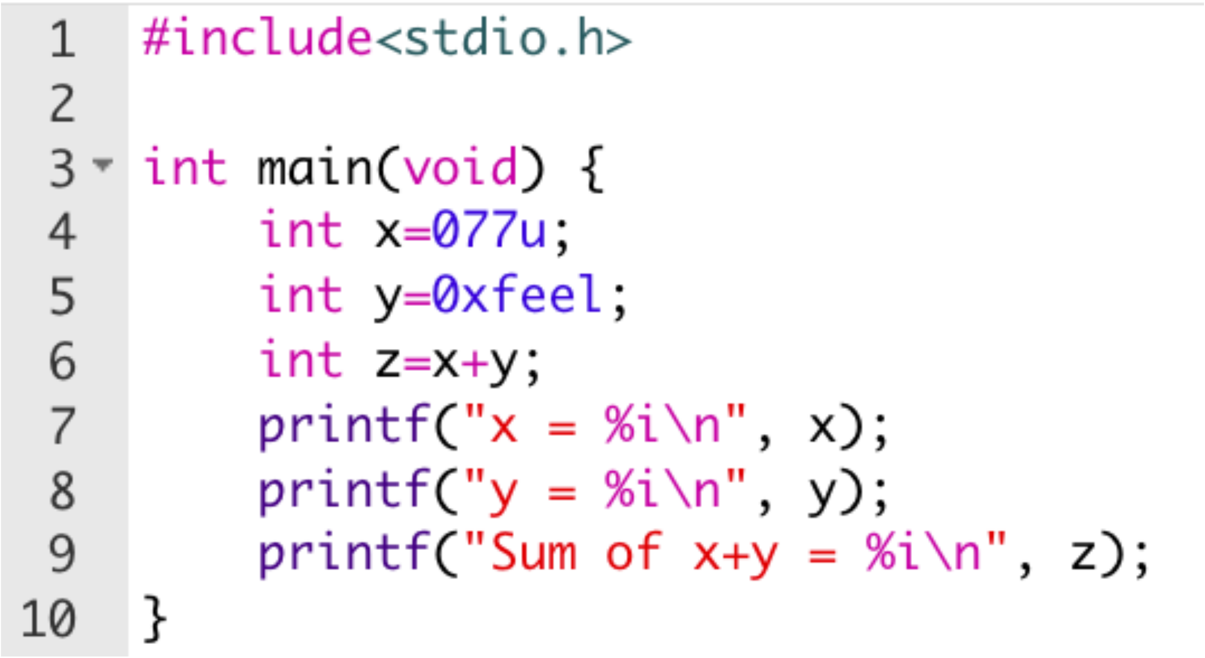
- Which one is illegal?



```
212         /* Legal */
215u        /* Legal */
0xFeeL      /* Legal */
078         /* Illegal: 8 is not an octal digit */
032UU       /* Illegal: cannot repeat a suffix */
```

Try this

- What does this program produce?



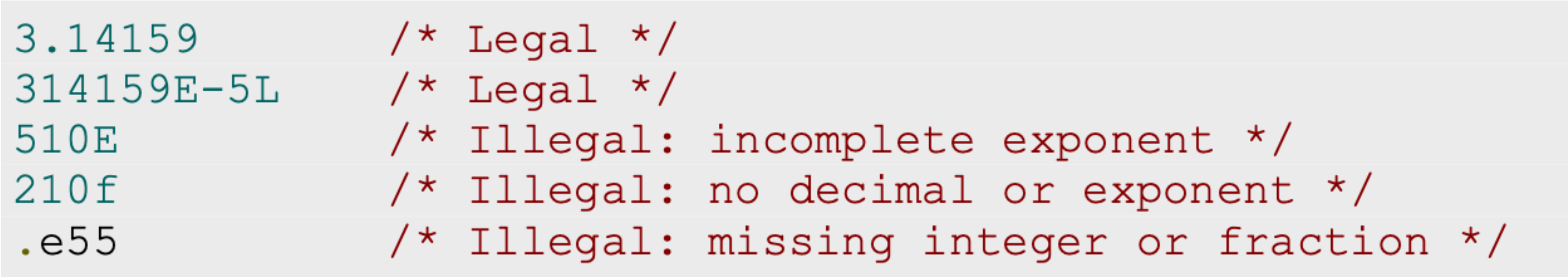
```c
#include<stdio.h>

int main(void) {
    int x=077u;
    int y=0xfeel;
    int z=x+y;
    printf("x = %i\n", x);
    printf("y = %i\n", y);
    printf("Sum of x+y = %i\n", z);
}
```

Try this: jdoodle.com/ia/IB5

Constants: Floating-point literals

- Decimal form

- Scientific notation form

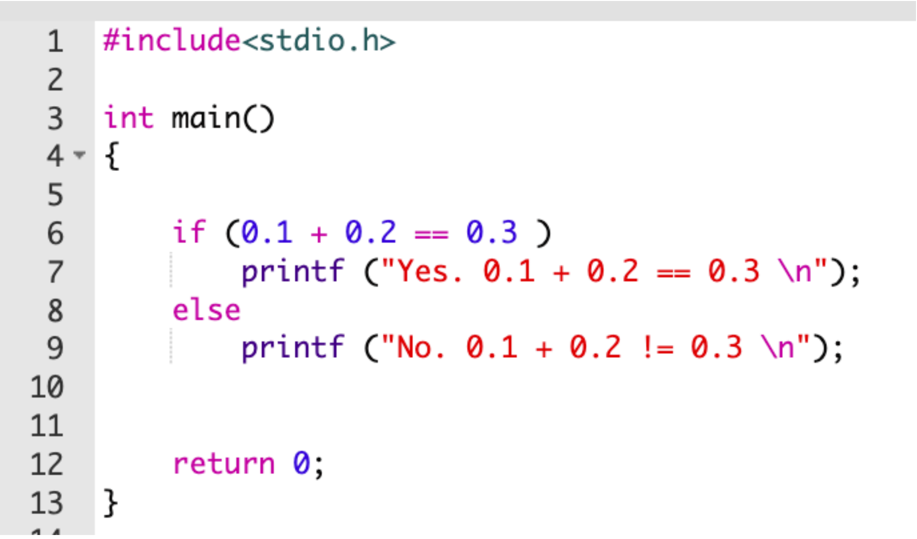


```
3.14159       /* Legal */
314159E-5L    /* Legal */
510E          /* Illegal: incomplete exponent */
210f          /* Illegal: no decimal or exponent */
.e55          /* Illegal: missing integer or fraction */
```

This kind of details is for your literature, not for the exam.

Question

- What does this program produce?

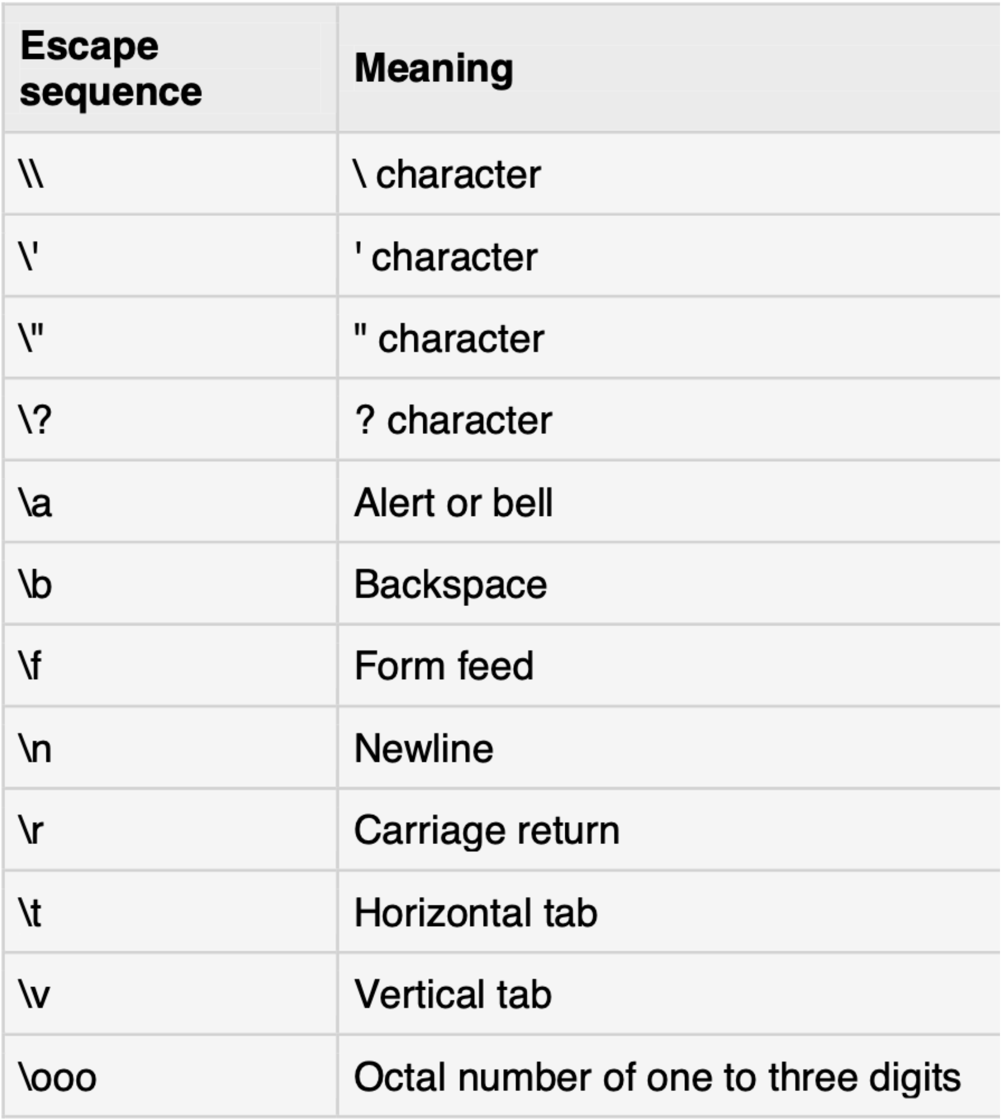


```c
#include<stdio.h>

int main()
{

    if (0.1 + 0.2 == 0.3 )
        printf ("Yes. 0.1 + 0.2 == 0.3 \n");
    else
        printf ("No. 0.1 + 0.2 != 0.3 \n");


    return 0;
}
```

jdoodle.com/ia/IB9

Constants: chars

- plain character (e.g., 'x'),
- escape sequence (e.g., ‘\t')
- universal character (e.g., '\u02C0').

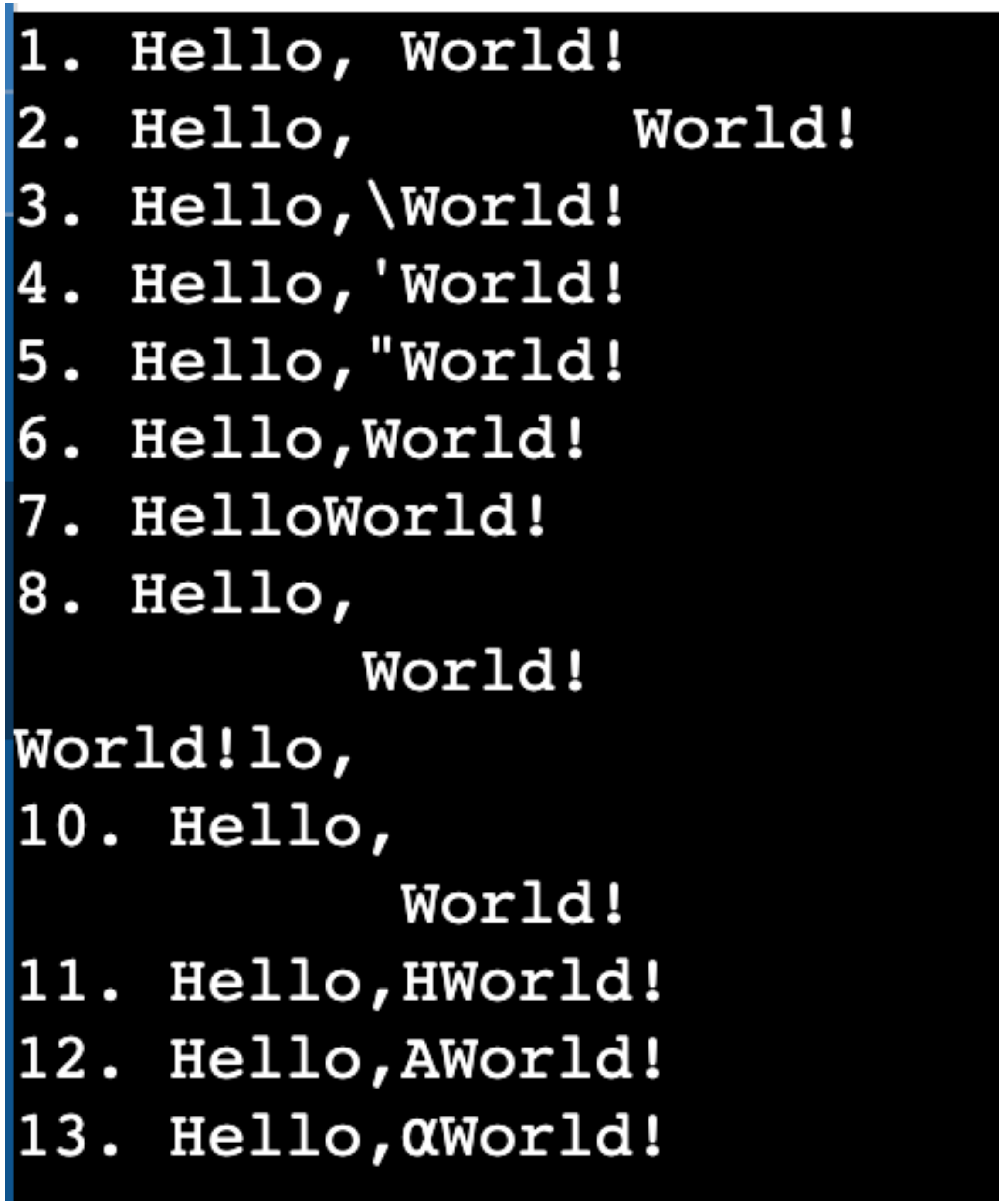


Escape sequence	Meaning
\\	\ character
\'	' character
\"	" character
\?	? character
\a	Alert or bell
\b	Backspace
\f	Form feed
\n	Newline
\r	Carriage return
\t	Horizontal tab
\v	Vertical tab
\ooo	Octal number of one to three digits

Try this

```
#include <stdio.h>

int main() {
    printf("1. Hello, World!\n");           // newline escape character
    printf("2. Hello,\tWorld!\n");          // tab escape character
    printf("3. Hello,\\World!\n");          // backslash escape character
    printf("4. Hello,\'World!\n");          // single quote escape character
    printf("5. Hello,\"World!\n");          // double quote escape character
    printf("6. Hello,\aWorld!\n");          // alert(bell) escape character
    printf("7. Hello,\bWorld!\n");          // backspace escape character
    printf("8. Hello,\fWorld!\n");          // form feed escape character
    printf("9. Hello,\rWorld!\n");          // carriage return escape character
    printf("10. Hello,\vWorld!\n");          // vertical tab escape character
    printf("11. Hello,\x48World!\n");        // hexadecimal number escape character
    printf("12. Hello,\101World!\n");        // octal number escape character
    printf("13. Hello,\u03B1World!\n");      // unicode escape character

    return 0;
}
```

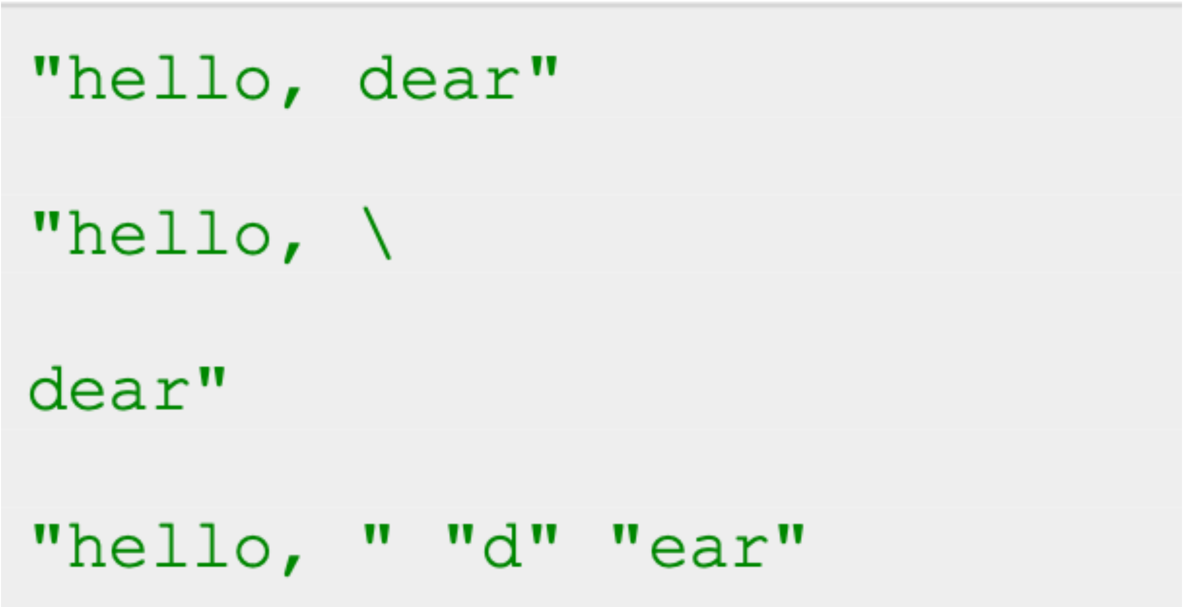
```c
#include <stdio.h>

int main() {
    printf("1. Hello, World!\n");
    printf("2. Hello,\tWorld!\n");
    printf("3. Hello,\\World!\n");
    printf("4. Hello,\'World!\n");
    printf("5. Hello,\"World!\n");
    printf("6. Hello,\aWorld!\n");
    printf("7. Hello,\bWorld!\n");
    printf("8. Hello,\fWorld!\n");
    printf("9. Hello,\rWorld!\n");
    printf("10. Hello,\vWorld!\n");
    printf("11. Hello,\x48World!\n");
character
    printf("12. Hello,\101World!\n");
    printf("13. Hello,\u03B1World!\n");

    return 0;
}
```



```
1. Hello, World!
2. Hello,       World!
3. Hello,\World!
4. Hello,'World!
5. Hello,"World!
6. Hello,World!
7. HelloWorld!
8. Hello,
         World!
World!lo,
10. Hello,
          World!
11. Hello,HWorld!
12. Hello,AWorld!
13. Hello,αWorld!
```

Constants: strings

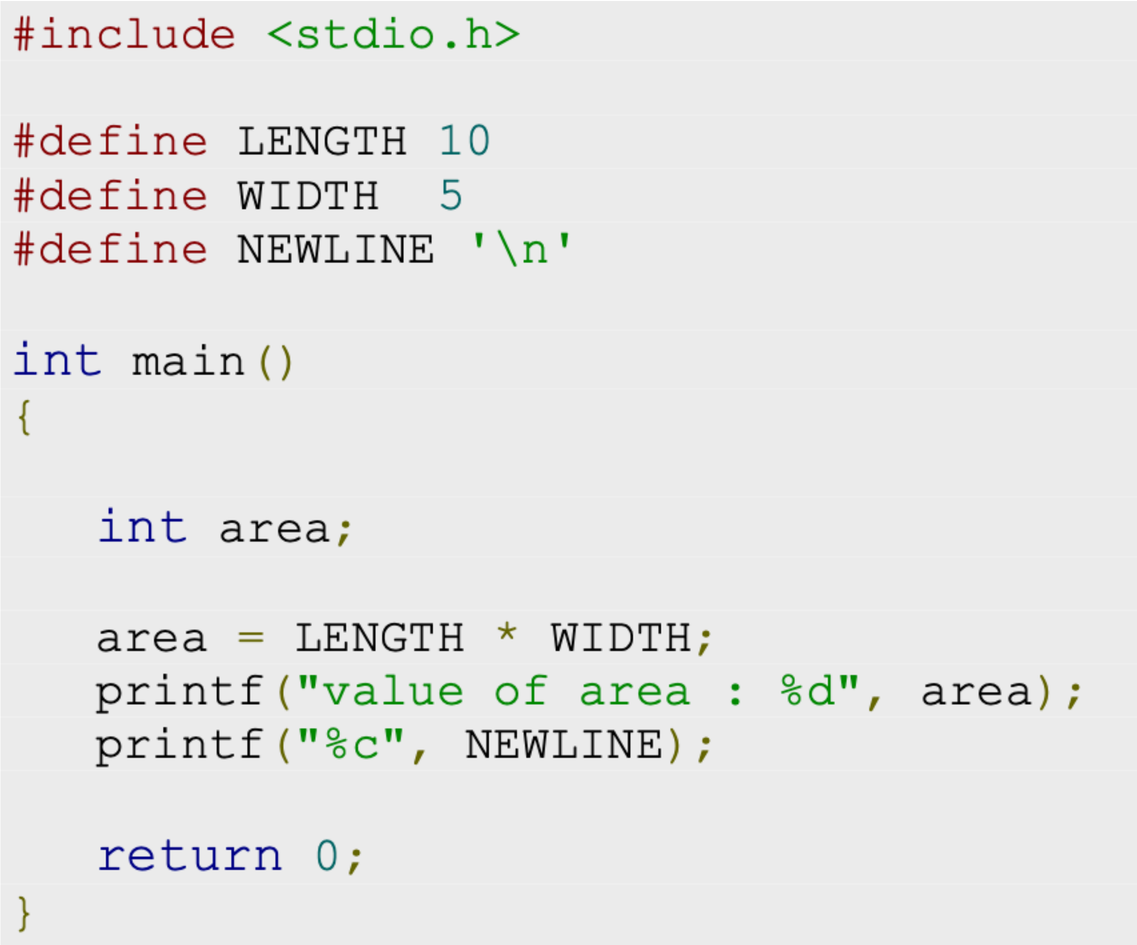
- strings = char sequences ending with \0
- break a long line into multiple lines = separate them using whitespaces
- All the three forms are identical



```
"hello, dear"

"hello, \

dear"

"hello, " "d" "ear"
```

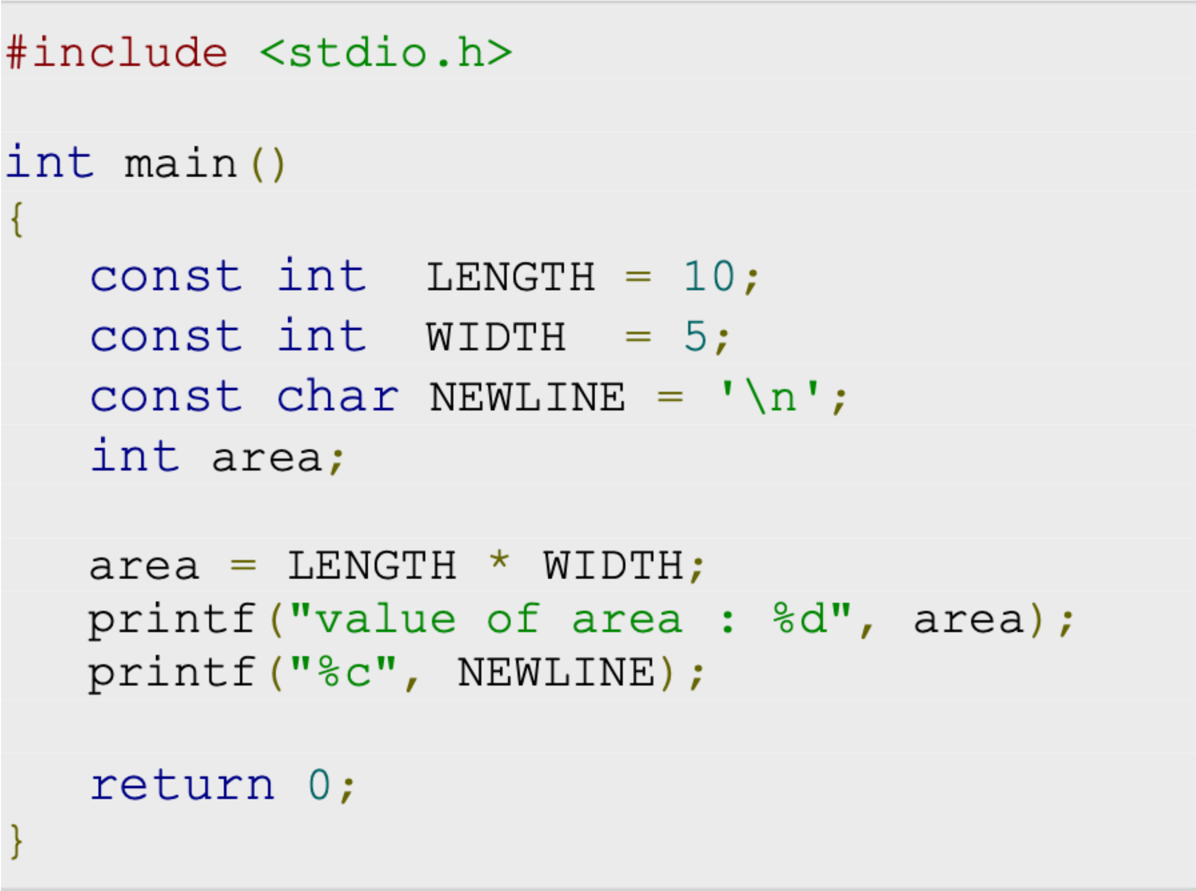
Defining Constants

Using #define preprocessor.



```c
#include <stdio.h>

#define LENGTH 10
#define WIDTH  5
#define NEWLINE '\n'

int main()
{

   int area;

   area = LENGTH * WIDTH;
   printf("value of area : %d", area);
   printf("%c", NEWLINE);

   return 0;
}
```

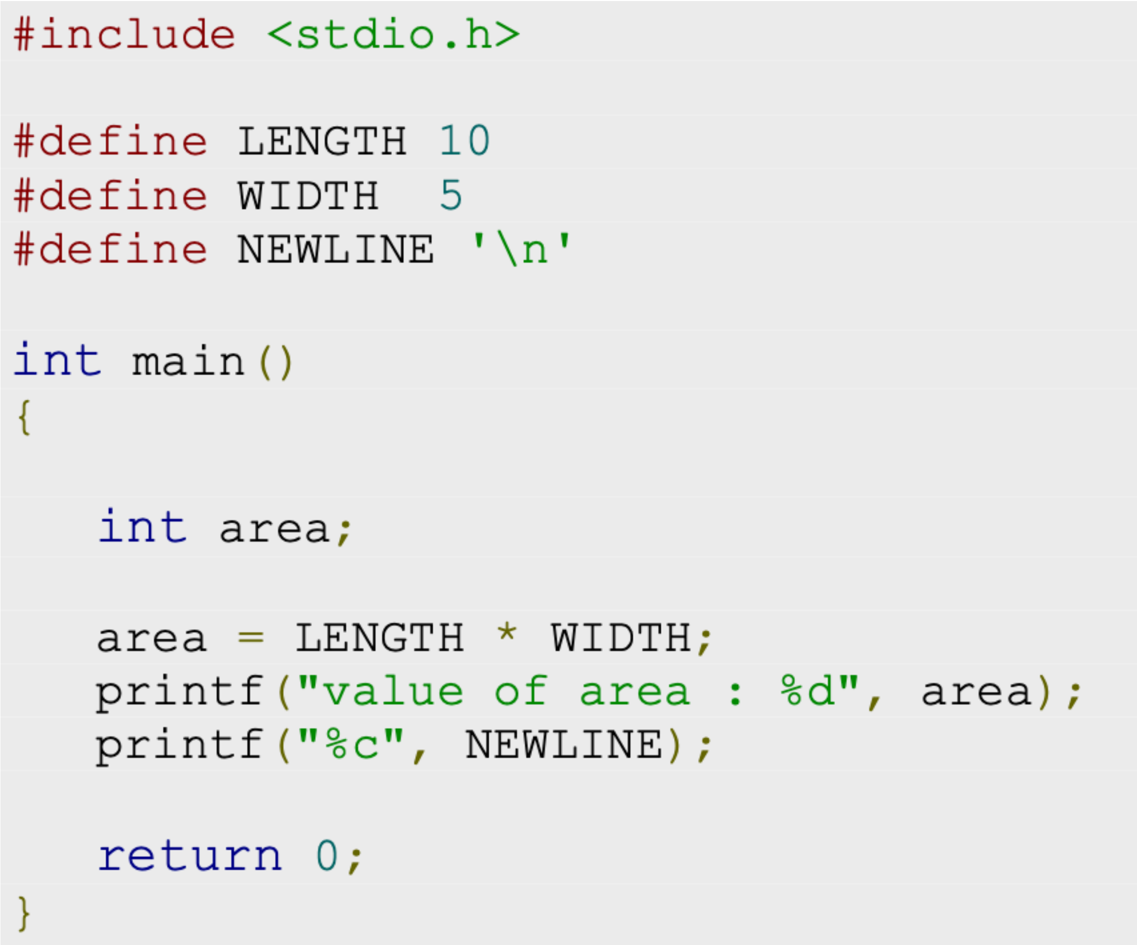
Using const preprocessor.



```c
#include <stdio.h>

int main()
{
   const int  LENGTH = 10;
   const int  WIDTH  = 5;
   const char NEWLINE = '\n';
   int area;

   area = LENGTH * WIDTH;
   printf("value of area : %d", area);
   printf("%c", NEWLINE);

   return 0;
}
```

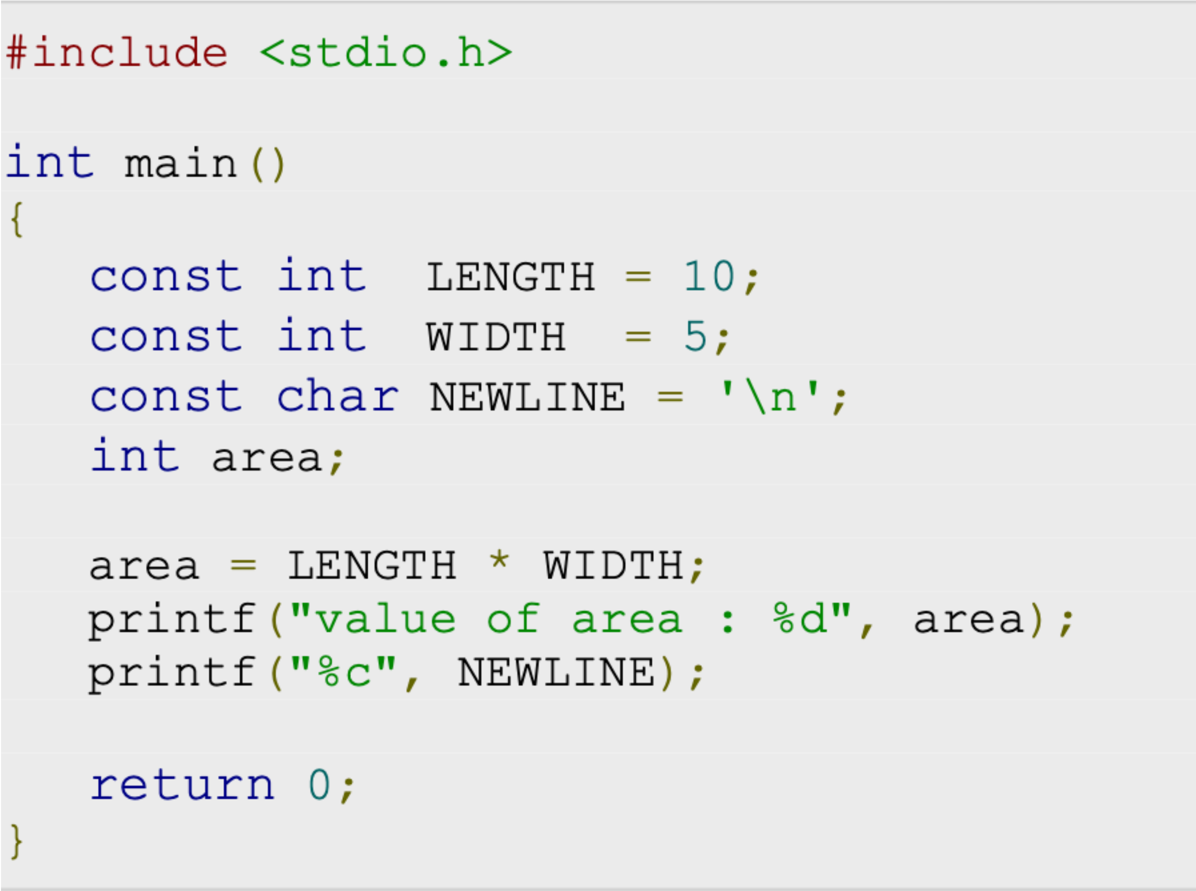
Defining Constants

Using #define preprocessor.



```c
#include <stdio.h>

#define LENGTH 10
#define WIDTH  5
#define NEWLINE '\n'

int main()
{

   int area;

   area = LENGTH * WIDTH;
   printf("value of area : %d", area);
   printf("%c", NEWLINE);

   return 0;
}
```

Using const keyword.



```c
#include <stdio.h>

int main()
{
   const int  LENGTH = 10;
   const int  WIDTH  = 5;
   const char NEWLINE = '\n';
   int area;

   area = LENGTH * WIDTH;
   printf("value of area : %d", area);
   printf("%c", NEWLINE);

   return 0;
}
```

#define LENGTH vs const int LENGTH ?

const int LENGTH vs extern int LENGTH vs. static int LENGTH ?

Storage classes

- auto:
- register:
- static:
- extern:

Storage classes

- auto: Variable allocated when the block in which they are defined is entered, and deallocated when it is exited.
- register: local variables that should be stored in a register instead of RAM.
- static: existence during the life-time
- extern: give a reference of a global variable that is visible to ALL the program files.

Auto

- auto: Variable allocated when the block in which they are defined is entered, and deallocated when it is exited.

```
void function() {
    auto int x = 0; // Here, `auto` is redundant because `x` is a local variable
    // ...
}
```

Register

- register: local variables that should be stored in a register instead of RAM.

```
#include <stdio.h>

int main() {
    register int counter;
    for(counter=0; counter<100000; counter++) {
        printf("%d\n", counter);
    }
    return 0;
}
```

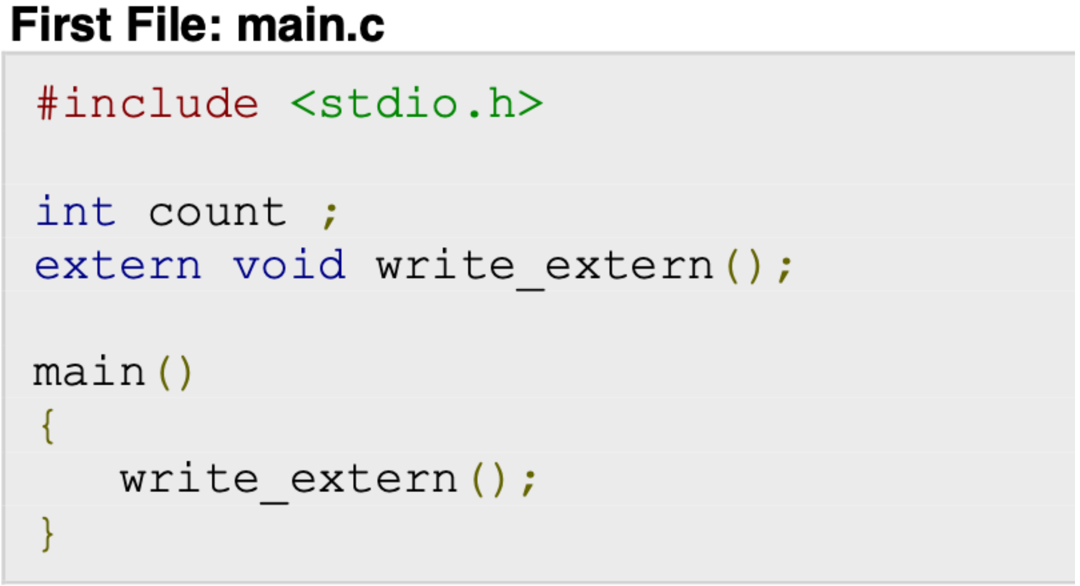
Static

- static: existence during the life-time

```
#include <stdio.h>

void increment() {
    static int count = 0;
    count++;
    printf("%d\n", count);
}

int main() {
    increment();  // prints 1
    increment();  // prints 2
    increment();  // prints 3
    return 0;
}
```

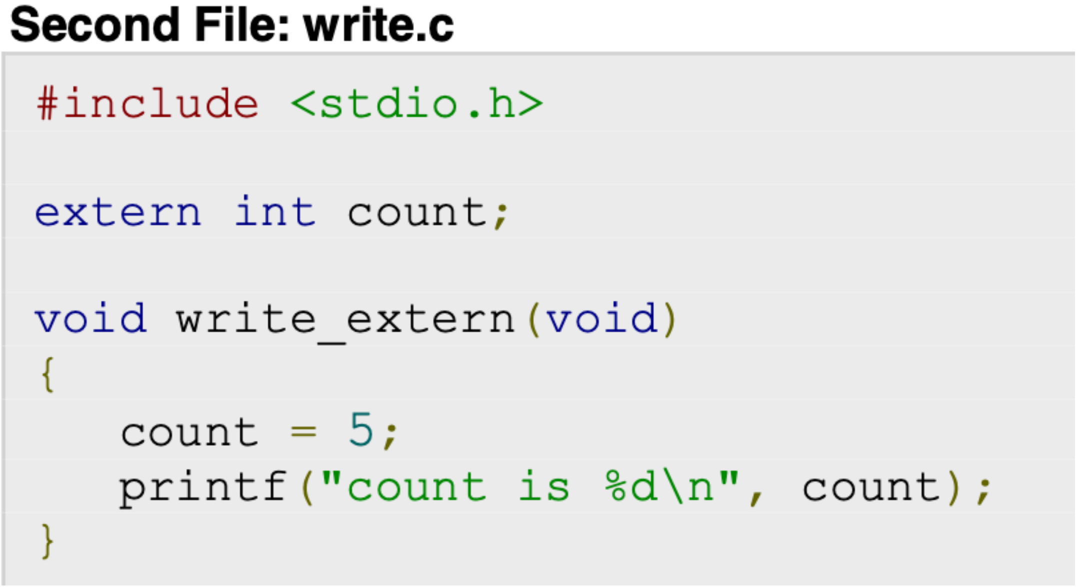
extern

- extern: give a reference of a global variable that is visible to ALL the program files.



First File: main.c

```
#include <stdio.h>

int count ;
extern void write_extern();

main()
{
   write_extern();
}
```



Second File: write.c

```
#include <stdio.h>

extern int count;

void write_extern(void)
{
   count = 5;
   printf("count is %d\n", count);
}
```

Lab exercise 1:
Implementing a Caesar Cipher in C

Introduction

- n 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.

Background

- **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.

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".

Lab exercise 2:
Sentence Title Case Verification in C

Problem

- 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.

Background

- 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.

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

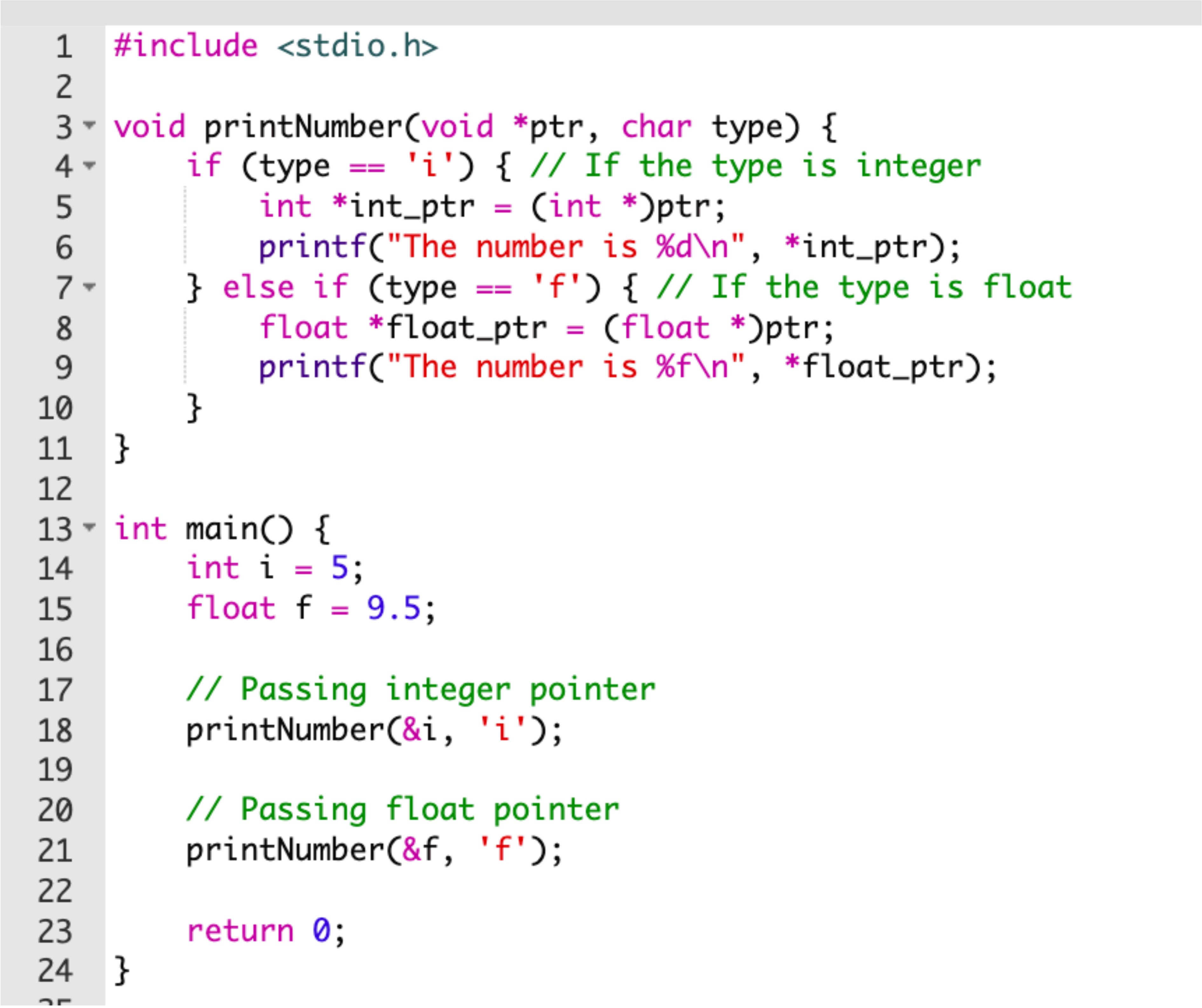
FANG Interview Question 1: const char* vs. char* const

- 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*?

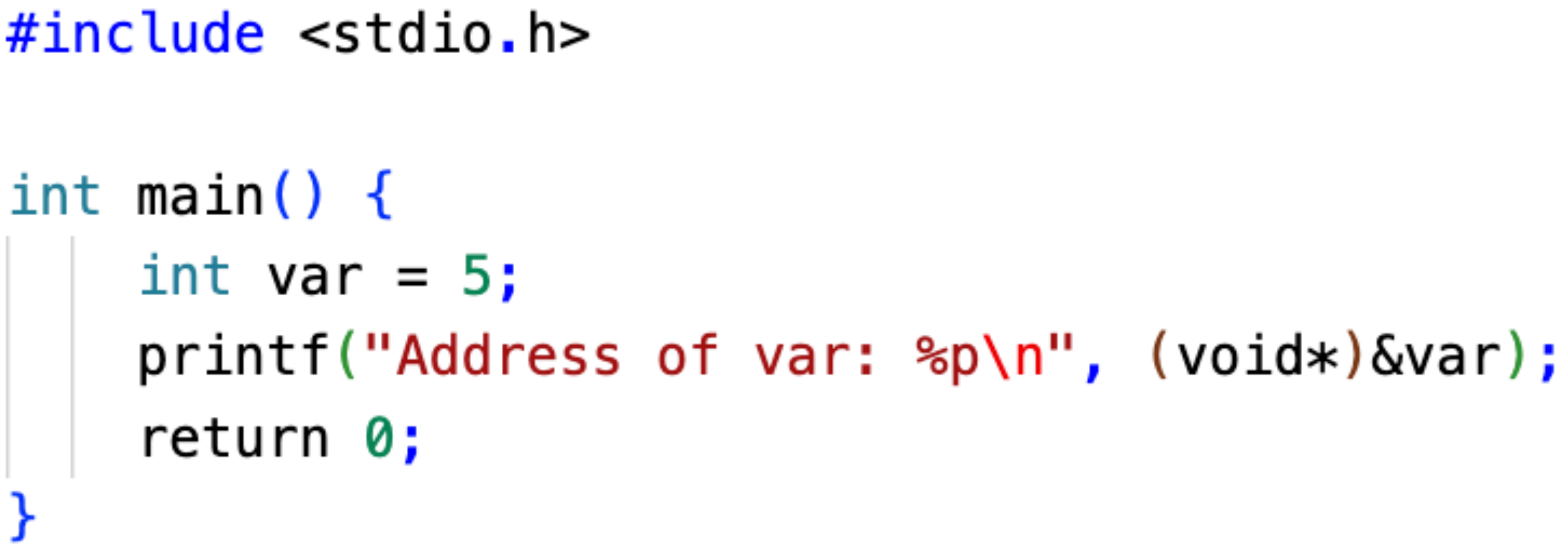
- generic pointer type
- raw address in memory
- can store the address of any object
- can be type-casted to any type of pointer
- Cannot be directly dereferenced. To access the object, you must first cast void * to a correct pointer type.



```c
#include <stdio.h>

void printNumber(void *ptr, char type) {
    if (type == 'i') { // If the type is integer
        int *int_ptr = (int *)ptr;
        printf("The number is %d\n", *int_ptr);
    } else if (type == 'f') { // If the type is float
        float *float_ptr = (float *)ptr;
        printf("The number is %f\n", *float_ptr);
    }
}

int main() {
    int i = 5;
    float f = 9.5;

    // Passing integer pointer
    printNumber(&i, 'i');

    // Passing float pointer
    printNumber(&f, 'f');

    return 0;
}
```

Address of (&)

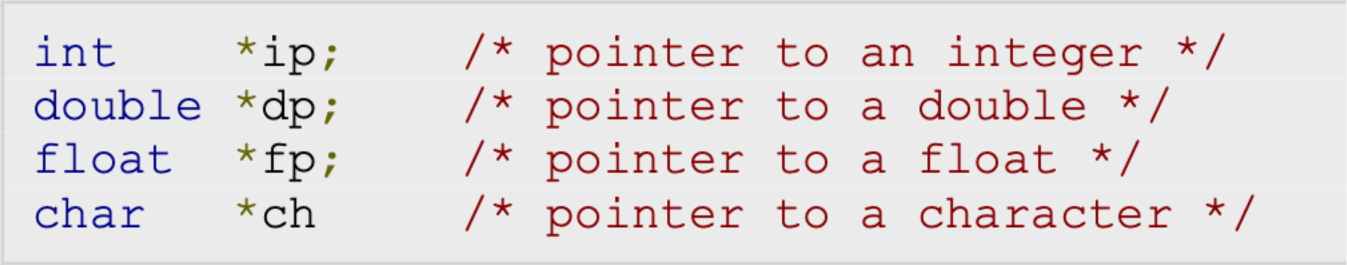
- To get the address of a variable, we use the ampersand (&) operator,



```c
#include <stdio.h>

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

Pointers (*)

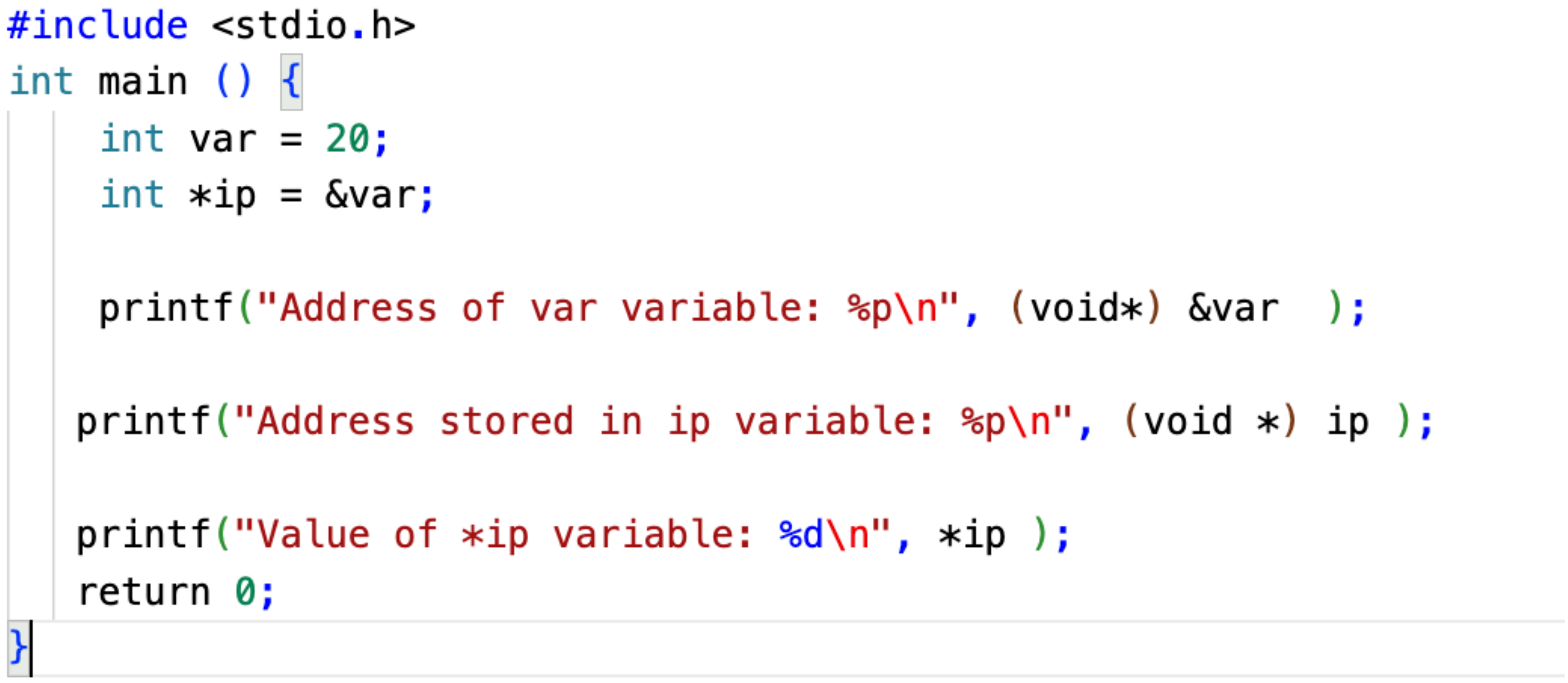
- A pointer is a variable whose value is an address
- To define a pointer, use **type *var-name;**



```
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;
}
```

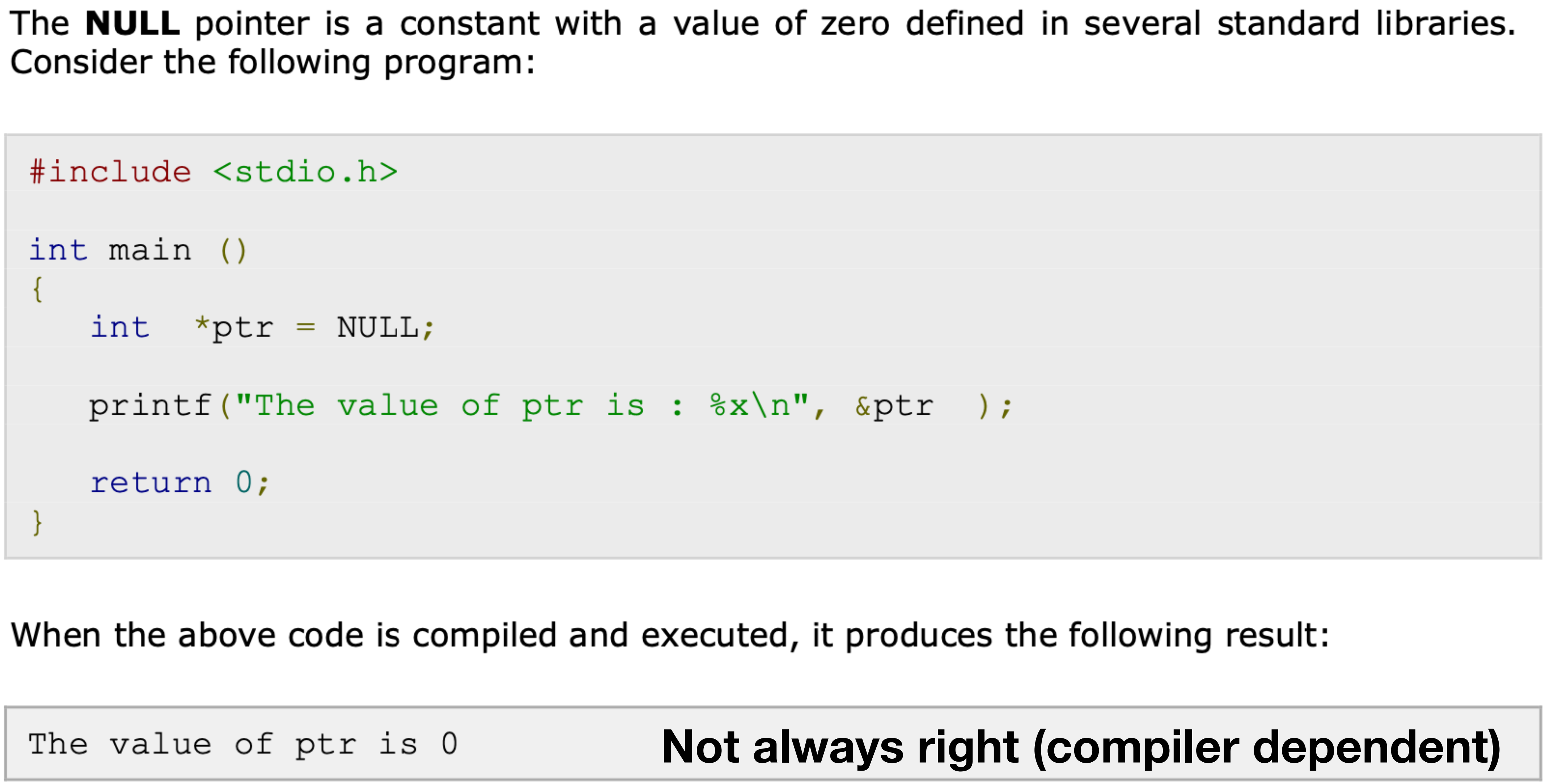
Question

- 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

Issue 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;
}
```

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

```
The value of ptr is 0
```

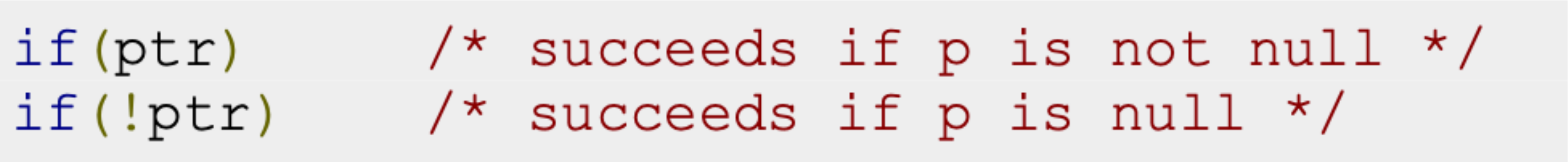
Not always right (compiler dependent)

Question: What will happen if we run this

```c
#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

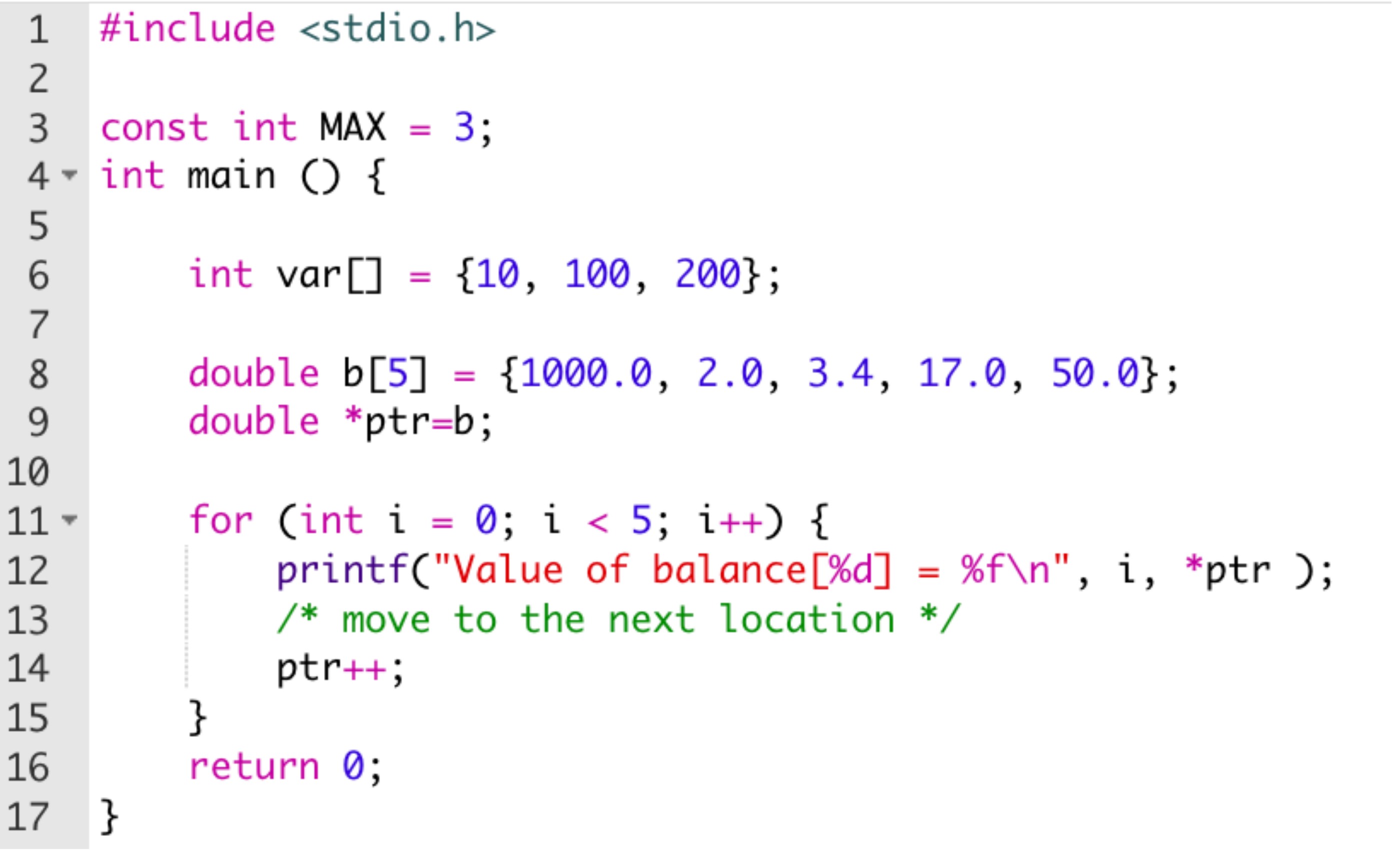


```
if(ptr)      /* succeeds if p is not null */
if(!ptr)     /* succeeds if p is null */
```

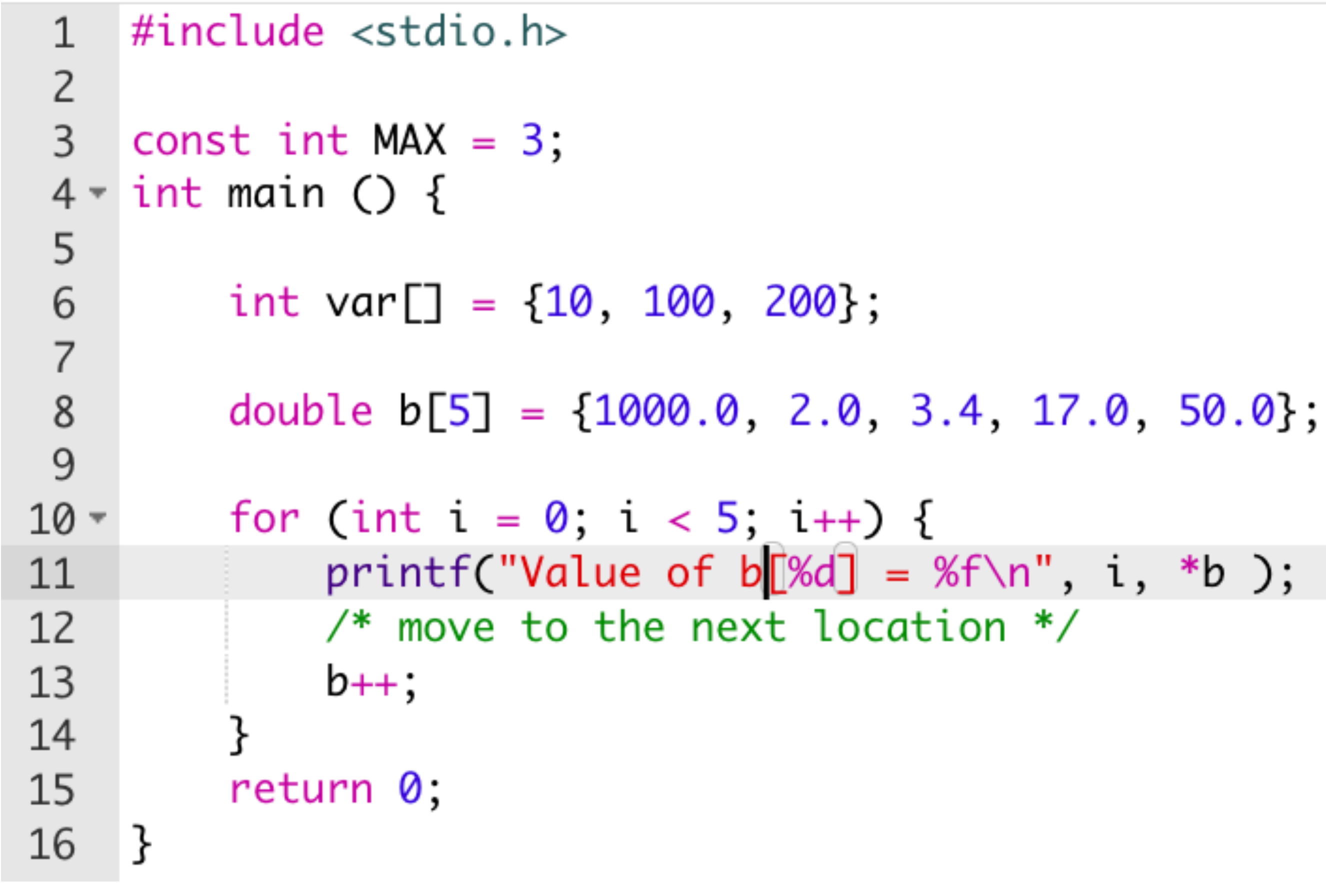
FANG Interview Question 3: Pointer arithmetics

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

FANG Interview Question 4: Which one will crash, and why?



```c
#include <stdio.h>

const int MAX = 3;
int main () {

    int var[] = {10, 100, 200};

    double b[5] = {1000.0, 2.0, 3.4, 17.0, 50.0};
    double *ptr=b;

    for (int i = 0; i < 5; i++) {
        printf("Value of balance[%d] = %f\n", i, *ptr );
        /* move to the next location */
        ptr++;
    }
    return 0;
}
```



```c
#include <stdio.h>

const int MAX = 3;
int main () {

    int var[] = {10, 100, 200};

    double b[5] = {1000.0, 2.0, 3.4, 17.0, 50.0};

    for (int i = 0; i < 5; i++) {
        printf("Value of b[%d] = %f\n", i, *b );
        /* move to the next location */
        b++;
    }
    return 0;
}
```

Arrays

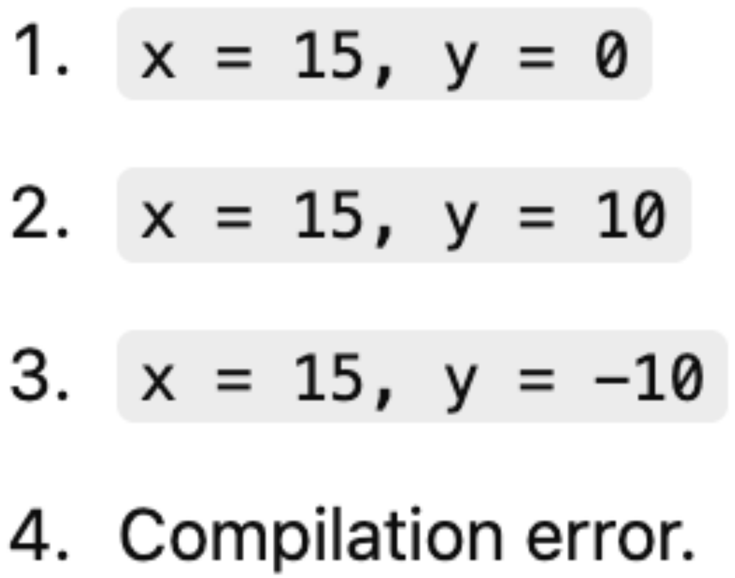
- Arrays are constant pointers
- int x[5];
- x++ would not work
- Use int * ptr = x; ptr++;

C Exercises

1

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

```
#include <stdio.h>

void trickyFunction(int *p1, int *p2) {
    *p1 = *p1 + *p2;
    p1 = p2;
    *p1 = *p1 - *p2;
}

int main() {
    int x = 5, y = 10;
    int *ptr1 = &x, *ptr2 = &y;

    trickyFunction(ptr1, ptr2);

    printf("x = %d, y = %d\n", x, y);
    return 0;
}
```



1. `x = 15, y = 0`
2. `x = 15, y = 10`
3. `x = 15, y = -10`
4. Compilation error.

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;
}
```