

# CS 330 & CS 332

## Final Exam Prep

C Programming Questions – Part 2

TRUE/ FALSE

1. Assuming `arr` is an array of integers, `sizeof(arr)` will return the number of elements in that array.

1. Assuming `arr` is an array of integers, `sizeof(arr)` will return the number of elements in that array.

**FALSE**

1. Assuming `arr` is an array of integers, `sizeof(arr)` will return the number of elements in that array.

# FALSE

To calculate the size of an array in C, we would need to write something like this:

```
sizeof(arr)/sizeof(arr[0])
```

2. Variables must be declared and defined at the same time.

2. Variables must be declared and defined at the same time.

**FALSE**

2. Variables must be declared and defined at the same time.

**FALSE**

We can declare variables before defining them in C programs, i.e.

```
int myint;  
myint = 12;
```



3. `string` is a supported data type in C.

3. `string` is a supported data type in C.

**FALSE**

3. `string` is a supported data type in C.

**FALSE**

In C, we use `char` arrays to store and represent strings of characters.

4. To print integers, we can use either the `%d` or the `%i` format specifiers.

4. To print integers, we can use either the `%d` or the `%i` format specifiers.

**TRUE**

4. To print integers, we can use either the `%d` or the `%i` format specifiers.

**TRUE**

Either of these format specifiers can be used to display integers.

5. `int 1value;` is a valid variable declaration.

5. `int 1value;` is a valid variable declaration.

**FALSE**



5. `int 1value;` is a valid variable declaration.

**FALSE**

While valid variable names may CONTAIN letters, underscores, or digits, they cannot START with a digit.

6. `bool` is NOT a supported data type in C.

6. `bool` is NOT a supported data type in C.

**TRUE**

6. `bool` is NOT a supported data type in C.

**TRUE**

To represent Booleans in C, we typically use integers assigned to 1 or 0 to indicate true or false, respectively.

7. `malloc()` assigns memory on the stack and returns a pointer to that memory.

7. `malloc()` assigns memory on the stack and returns a pointer to that memory.

**FALSE**

7. `malloc()` assigns memory on the stack and returns a pointer to that memory.

**FALSE**

`malloc()` assigns memory from the HEAP, then returns a pointer to that memory.

8. The C compilation process goes as follows:

preprocessor => compiler => assembler => linker



8. The C compilation process goes as follows:

preprocessor => compiler => assembler => linker

**TRUE**

8. The C compilation process goes as follows:

preprocessor => compiler => assembler => linker

# TRUE

preprocessor - expands the source code.

compiler - converts pre-processed code into assembly.

assembler - converts assembly code into object code.

linker - combines the object code from our program with the object code of C libraries and other files.

9. Any code written with an if-else statement can be rewritten using switch and vice-versa.

9. Any code written with an if-else statement can be rewritten using switch and vice-versa.

**FALSE**

9. Any code written with an if-else statement can be rewritten using switch and vice-versa.

**FALSE**

If-else statements branch on whether condition(s) is/are true or false. Switch statements branch on the equality of a variable (int or char) with enumerated cases.

	if-else	switch
Definition	Depending on the condition in the 'if' statement, 'if' and 'else' blocks are executed.	The user will decide which statement is to be executed.
Expression	It contains either logical or equality expression.	It contains a single expression which can be either a character or integer variable.
Evaluation	It evaluates all types of data, such as integer, floating-point, character or Boolean.	It evaluates either an integer, or character.
Sequence of Execution	First, the condition is checked. If the condition is true then 'if' block is executed otherwise 'else' block	It executes one case after another till the break keyword is not found, or the default statement is executed.
Default execution	If the condition is not true, then by default, else block will be executed.	If the value does not match with any case, then by default, default statement is executed.
Editing	Editing is not easy in the 'if-else' statement.	Cases in a switch statement are easy to maintain and modify. Therefore, we can say that the removal or editing of any case will not interrupt the execution of other cases.
Speed	If there are multiple choices implemented through 'if-else', then the speed of the execution will be slow.	If we have multiple choices then the switch statement is the best option as the speed of the execution will be much higher than 'if-else'.

10. Functions in C must always include a **return** statement.

10. Functions in C must always include a **return** statement.

**FALSE**



10. Functions in C must always include a **return** statement.

**FALSE**

If a function returns **void**, a **return** statement is not needed.

MULTIPLE CHOICE

11. Every C program must contain which of the following:

- A. `#include <stdio.h>`
- B. `#include <stdlib.h>`
- C. `main()`
- D. None of the above.

11. Every C program must contain which of the following:

- A. `#include <stdio.h>`
- B. `#include <stdlib.h>`
- C. `main()` **CORRECT**
- D. None of the above.

12. What does `!=` do?

- A. Logical NOT
- B. Difference (inequality)
- C. Unassign
- D. Equality

12. What does != do?

- A. Logical NOT
- B. Difference (inequality) CORRECT
- C. Unassign
- D. Equality

13. What does **const** do?

- A. Specifies that a value will not be changed.
- B. Makes immutable data types mutable.
- C. Precedes pointer assignment.
- D. Creates a user defined data type.

13. What does **const** do?

CORRECT

- A. Specifies that a value will not be changed.
- B. Makes immutable data types mutable.
- C. Precedes pointer assignment.
- D. Creates a user defined data type.



14. What will the following program return?

```
int add1(int n) {  
    return n + 1;  
}
```

```
int main() {  
    int n = 5;  
    add1(n);  
    add1(n);  
    return n;  
}
```

A. 5

B. 6

C. 7

D. compiler error

14. What will the following program return?

```
int add1(int n) {  
    return n + 1;  
}
```

```
int main() {  
    int n = 5;  
    add1(n);  
    add1(n);  
    return n;  
}
```

A. 5 CORRECT

B. 6

C. 7

D. compiler error

15. What will the following program print?

```
main() {  
    int x = 1, y = 0, z = 5;  
    int a = x && y || z++;  
    printf("%d", z);  
}
```

A. 0  
B. 5

C. 6  
D. error

15. What will the following program print?

```
main() {  
    int x = 1, y = 0, z = 5;  
    int a = x && y || z++;  
    printf("%d", z);  
}
```

A. 0  
B. 5

C. 6 CORRECT  
D. error

16. Which of the following are correct pointer initializations?

A. `int ptr1* = &a;`

B. `int ptr2** = NULL;`

C. A and B.

D. None of the above.

16. Which of the following are correct pointer initializations?

A. `int ptr1* = &a;`

B. `int ptr2** = NULL;`

C. A and B. CORRECT

D. None of the above.

17. What will the following program return?

```
void add1(int *n) {  
    *n += 1;  
}  
  
int main() {  
    int n = 3;  
    int *ptr = &n;  
    add1(ptr);  
    add1(ptr);  
    return n;  
}
```

A. 3  
B. 4  
C. 5  
D. compiler error

17. What will the following program return?

```
void add1(int *n) {  
    *n += 1;  
}
```

```
int main() {  
    int n = 3;  
    int *ptr = &n;  
    add1(ptr);  
    add1(ptr);  
    return n;  
}
```

A. 3

B. 4

C. 5 CORRECT

D. compiler error



18. How do you access members of a struct via pointer?

- A. `mystruct->mem`
- B. `mystruct.mem`
- C. `mystruct[mem]`
- D. All of the above

18. How do you access members of a struct via pointer?

- A. `mystruct->mem` **CORRECT**
- B. `mystruct.mem`
- C. `mystruct[mem]`
- D. All of the above

19. Which of the following type-casting is accepted in C?

- A. Implicit type conversion
- B. Explicit type conversion
- C. Both
- D. None of the above

19. Which of the following type-casting is accepted in C?

- A. Implicit - done by compiler,  $\text{int} + \text{float} = \text{float}$
- B. Explicit - type casting, i.e.  $(\text{type})\text{var}$
- C. Both CORRECT
- D. None of the above

20. Which of these function prototypes do NOT allow an array to be passed?

- A. `void myfun(int arr[], int size)`
- B. `void myfun(int arr[4], int size)`
- C. `void myfun(int *arr, int size)`
- D. `void myfun(int arr, int size)`

20. Which of these function prototypes do NOT allow an array to be passed?

- A. `void myfun(int arr[], int size)`
- B. `void myfun(int arr[4], int size)`
- C. `void myfun(int *arr, int size)`
- D. `void myfun(int arr, int size)`

CORRECT

# Thank you for coming!

Please write your blazerid on the  
whiteboard on your way out.