CS 330 & CS 332
Final Exam Prep

C Programming Questions – Part 1

TRUE/ FALSE

1. A preprocessor command makes your code compile faster.

1. A preprocessor command makes your code compile faster.

FALSE

1. A preprocessor command makes your code compile faster.

FALSE

Preprocessor commands have NO BEARING on compilation speed.

What do preprocessor commands do?

What do preprocessor commands do?

A preprocessor is a text substitution tool the compiler uses before performing the actual compilation. There are several commands, but the ones we've used most often are `#include` and `#define`.

2. A compiler converts a high-level language to executable machine code.

2. A compiler converts a high-level language to executable machine code.

TRUE

2. A compiler converts a high-level language to executable machine code.

TRUE

A compiler translates source code into machine-language instructions. Our C compilers do this by way of first converting source files into assembly, then bytecode.

3. A library is a source file that contains ready-made functions.

3. A library is a source file that contains ready-made functions.

TRUE

3. A library is a source file that contains ready-made functions.

TRUE

This is exactly what a C library is!

4. C functions cannot call themselves.

4. C functions cannot call themselves.

FALSE

4. C functions cannot call themselves.

FALSE

C functions CAN call themselves!
The language supports recursion.

5. A `struct` is a user defined data type.

5. A `struct` is a user defined data type.

TRUE

5. A `struct` is a user defined data type.

TRUE

What are the other user defined data types in C?

What are the user defined data types in C?

- `struct`
- `union`

  a collection of different data types, but only one member can contain a value.
- `typedef`

  creates an alias (new name) for a data type that already exists.
- `enum`

  consists of a set of named values.

6. Converting a variable from an `int` to a `float` will never affect its value.

6. Converting a variable from an `int` to a `float` will never affect its value.

FALSE

6. Converting a variable from an `int` to a `float` will never affect its value.

FALSE

While both `int`s and `float`s are 4 bytes in size, a large enough `int` would get truncated when converted to a `float`, because not all its width is used to represent a whole number.

7. `while` loops are faster than `for` loops.

7. `while` loops are faster than `for` loops.

FALSE

7. `while` loops are faster than `for` loops.

FALSE

`while` loops are NOT faster than `for` loops.

8. A `do-while` loop will always execute at least once.

8. A `do-while` loop will always execute at least once.

TRUE

8. A `do-while` loop will always execute at least once.

TRUE

A `do-while` loop will execute its statement first, BEFORE checking the loop condition.

9. C is an object-oriented language.

9. C is an object-oriented language.

FALSE

9. C is an object-oriented language.

FALSE

While `structs` allow us to implement some OOP principles in the language, C is NOT object-oriented.

10. C is a low-level language.

10. C is a low-level language.

FALSE

10. C is a low-level language.

FALSE

While C gets closer to the wire than the languages we may have learned prior, (Python, Java), C is itself a high-level language.

11. The **&** and **&&** operators are functionally equivalent

11. The **&** and **&&** operators are functionally equivalent

FALSE

11. The **&** and **&&** operators are functionally equivalent

FALSE

& is the bitwise and operator and **&&** is the logical and operator.

12. The size of a pointer is always 8 bytes.

12. The size of a pointer is always 8 bytes.

TRUE

12. The size of a pointer is always 8 bytes.

TRUE

Regardless of the data type to which it is pointing, a pointer is always 8 bytes wide.

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

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

FALSE

13. 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])
```

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

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

FALSE

14. 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;
```

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

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

FALSE

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

FALSE

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

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

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

TRUE

16. 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.

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

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

FALSE

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

FALSE

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

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

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

TRUE

18. `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.

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

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

FALSE

19. `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.

20. The C compilation process goes as follows:

preprocessor => compiler => assembler => linker

20. The C compilation process goes as follows:

preprocessor => compiler => assembler => linker

TRUE

20. 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.

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

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

FALSE

21. 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.

https://www.javatpoint.com/if-else-vs-switch	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'.

22. Functions in C must always include a `return` statement.

22. Functions in C must always include a `return` statement.

FALSE

22. Functions in C must always include a `return` statement.

FALSE

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

MULTIPLE CHOICE

13. Which standard library includes the `printf()` and `scanf()` functions?

A. `<time.h>`
B. `<stdlib.h>`
C. `<stdio.h>`
D. `<printer.h>`

13. Which standard library includes the `printf()` and `scanf()` functions?

A. `<time.h>`
B. `<stdlib.h>`
C. `<stdio.h>` CORRECT
D. `<printer.h>`

14. What will the following program print?

```
main {
    float f = 9.45;
    int i = f;
    i += 0.55;
    f = i;
    printf("%f", f);
}
```

A. 9.0
B. 10.0
C. 9.55
D. 1.0

14. What will the following program print?

```
main {
    float f = 9.45;
    int i = f;
    i += 0.55;
    f = i;
    printf("%f", f);
}
```

A. 9.0 CORRECT
B. 10.0
C. 9.55
D. 1.0

15. What is the correct format specifier to print characters?

A. `%d`
B. `\c`
C. `%c`
D. `%lf`

15. What is the correct format specifier to print characters?

A. `%d`
B. `\c`
C. `%c` CORRECT
D. `%lf`

16. What will the following program print?

```
main {
    int i;
    for (i = 0; i < 10; i++)
        i += 2;
    printf("%d", i);
}
```

A. 9
B. 10
C. 11
D. 12

16. What will the following program print?

```
main {
    int i;
    for (i = 0; i < 10; i++)
        i += 2;
    printf("%d", i);
}
```

A. 9
B. 10
C. 11
D. 12
CORRECT

17. Between `gets()` and `fgets()`, which is the safer function?

A. `gets()`
B. `fgets()`
C. They are equally safe
D. They are equally unsafe

17. Between `gets()` and `fgets()`, which is the safer function?

A. `gets()`
B. `fgets()` CORRECT
C. They are equally safe
D. They are equally unsafe

18. What will the following program print?

```
main {
    int a, b;
    a = b = 50;
    b /= 2;
    a *= 2;
    printf("%d", ++a + b--);
}
```

A. 126
B. 125
C. 100
D. error

18. What will the following program print?

```
main {
    int a, b;
    a = b = 50;
    b /= 2;
    a *= 2;
    printf("%d", ++a + b--);
}
```

CORRECT

A. 126

B. 125

C. 100

D. error

19. Which method is used to convert an integer to a char/string data type?

A. `atoi()`
B. `itoa()`
C. `itos()`
D. `ctoi()`

19. Which method is used to convert an integer to a char/string data type?

A. `atoi()`
B. `itoa()` CORRECT
C. `itos()`
D. `ctoi()`

20. What will the following program print?

```
main {
    printf("%d", ((3/4) * 60) + 14);
}
```

A. 59
B. 0
C. 14
D. 45

20. What will the following program print?

```
main {
    printf("%d", ((3/4) * 60) + 14);
}
```

A. 59
B. 0
C. 14 CORRECT
D. 45

21. Below is a list of different variables. Which option lists these variables by descending size?*

```
char s[5];
int i;
long l;
struct mystruct {
    char x;
    char y;
    int z;
} STRUCT;
char c = '\n';
```

A. s, STRUCT, i, l, c

B. l, s, STRUCT, i, c

C. STRUCT, l, s, i, c

D. l, STRUCT, s, i, c

*Assume that the compiler is NOT adding padding to align data.

21. Below is a list of different variables. Which option lists these variables by descending size?*

```
char s[5];
int i;
long l;
struct mystruct {
    char x;
    char y;
    int z;
} STRUCT;
char c = '\n';
```

A. s, STRUCT, i, l, c

B. l, s, STRUCT, i, c

C. STRUCT, l, s, i, c
CORRECT

D. l, STRUCT, s, i, c

*Assume that the compiler is NOT adding padding to align data.

22. If we have some variable `int var`, `&var` would give us:

A. The address of `var`.
B. The data type of `var`.
C. The size of `var` (in bytes).
D. The value at the location of `var`.

22. If we have some variable `int var`, `&var` would give us:

A. The address of `var`. CORRECT
B. The data type of `var`.
C. The size of `var` (in bytes).
D. The value at the location of `var`.

23. If we have some variable `int var`, `*var` would give us:

A. The address of `var`.
B. The data type of `var`.
C. The size of `var` (in bytes).
D. The value at the location of `var`.

23. If we have some variable `int var`, `*var` would give us:

A. The address of `var`.
B. The data type of `var`.
C. The size of `var` (in bytes).
D. The value at the location of `var`. CORRECT

24. Given the following code, what would the output of `ptr2` be?

```
int a = 5;
int *ptr1 = &a;
int **ptr2 = &ptr1;
```

A. 5
B. The address of `ptr1`
C. The address of `a`
D. The value of `ptr1`

24. Given the following code, what would the output of `ptr2` be?

```
int a = 5;
int *ptr1 = &a;
int **ptr2 = &ptr1;
```

A. 5
B. The address of `ptr1` CORRECT
C. The address of `a`
D. The value of `ptr1`

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

* original code from question 10 on https://stackhowto.com/100-multiple-choice-questions-in-c-programming-part-7/

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

* original code from question 10 on https://stackhowto.com/100-multiple-choice-questions-in-c-programming-part-7/

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

* from question 8 on https://stackhowto.com/100-multiple-choice-questions-in-c-programming-part-8/

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

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

* from question 8 on https://stackhowto.com/100-multiple-choice-questions-in-c-programming-part-8/

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)`

* Based on question 1 from https://stackhowto.com/100-multiple-choice-questions-in-c-programming-part-9/

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

* Based on question 1 from https://stackhowto.com/100-multiple-choice-questions-in-c-programming-part-9/

Thank you for coming!

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