CS 330 & CS 332 Midterm 1 Prep

C Programming Questions

TRUE/FALSE

1. Assembly is a low-level language.

1. Assembly is a low-level language.

TRUE

Assembly provides direct instructions to the processor and registers.

2. Larger storage devices are faster and costlier, while smaller storage devices are slower and cheaper.

2. Larger storage devices are faster and costlier, while smaller storage devices are slower and cheaper.

FALSE

The inverse is true, registers and caches for example are faster and cheaper than main memory or local disks, which are slower and more expensive

3. Operating Systems provide uniform mechanisms for manipulating low-level hardware through abstraction.

3. Operating Systems provide uniform mechanisms for manipulating low-level hardware through abstraction.

TRUE

Files are an abstraction of I/O devices. Virtual Memory is an abstraction for main memory and I/O devices. Processes are abstractions for processor, main memory, and I/O devices

4. Computers use base ten to represent numbers.

4. Computers use base ten to represent numbers.

FALSE

This would be very difficult for computers to store; computers use binary to represent numbers.

5. The & and && operators are functionally equivalent.

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

FALSE

"**&**" is the bit-wise AND operator, while "**&&**" is the logical AND operator.

6. The `int` and `float` data types are the same size.

6. The `int` and `float` data types are the same size.

TRUE

Both data types are 4 bytes wide.

7. Casting a value stored as an `int` to a `float` will never affect the value.

7. Casting a value stored as an `int` to a `float` will never affect the value.

FALSE

`float`s cannot represent values as large or as small as `int`s can. For values that are large or small enough, converting an int to a float might truncate it.

8. C is a case sensitive language.

8. C is a case sensitive language.

TRUE

9. When performing arithmetic conversions implicitly, the smaller data type will always get converted to the larger of data types (eg. `int * double = double`).

9. When performing arithmetic conversions implicitly, the smaller data type will always get converted to the larger of data types (eg. `int * double = double`).

TRUE

10. Header files (**.h**) contain function definitions that get added to the source code when the preprocessor commands are run.

10. Header files (`.h`) contain function definitions that get added to the source code when the preprocessor commands are run.

FALSE

header files contain function PROTOTYPES. The function definitions exist in the `.lib`, or library file and are added in the linking stage of compilation.

11. `scanf()` accepts some string of format specifiers and some number memory addresses.

11. `scanf()` accepts some string of format specifiers and some number memory addresses.

TRUE

`scanf()` reads data from standard input and stores it at the given memory addresses.

12. Logic written using `if else` statements can always be expressed using `switch` statements instead.

12. Logic written using `if else` statements can always be expressed using `switch` statements instead.

FALSE

`switch` statements check exact equality, unlike `if-else` statements which can evaluate expressions.

13. `do-while` loops will always run at least once.

13. `do-while` loops will always run at least once.

TRUE

`do-while` executes some body of work before checking the loop condition, `while` loops check the condition first before doing any work.

14. Pointers are always 8 bytes wide.

14. Pointers are always 8 bytes wide.

TRUE

Pointers can point to data of any type, but since they store addresses themselves, pointers are always 8 bytes.

15. You cannot use double pointers in C.

15. You cannot use double pointers in C.

FALSE

You can absolutely use double pointers in C!

MULTIPLE CHOICE

16. Which is the correct way to declare a variable in C?

A. `c;`
B. `char c;`
C. `string c;`
D. `char c`

16. Which is the correct way to declare a variable in C?

A. `c;`

B. `char c;` CORRECT

C. `string c;`

D. `char c`

17. What will the following program print?

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

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

17. What will the following program print?

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

A. 59

B. 0

C. 14 CORRECT

D. 45

19. What will the following program print?

```
#include <stdio.h>
int main(void) {
    int a = 4, b = 2;
    a += 6;
    a *= b + 3;
    a -= b + 8;
    a /= b;
    a %= b + 1;
    printf("Num = %d\n", a);
    return 0;
}
```

A. 0 B. 1 C. 2 D. 3

19. What will the following program print?

```
#include <stdio.h>
int main(void) {
    int a = 4, b = 2;
    a += 6;
    a *= b + 3;
    a -= b + 8;
    a /= b;
    a %= b + 1;
    printf("Num = %d\n", a);
    return 0;
}
```

A. 0 B. 1 C. 2 D. 3

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

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

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

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

21. Given the following code, what will this program print?

```
int main() {
    int a = 5;
    int *ptr1 = &a;
    int **ptr2 = &ptr1;
    printf("%x", ptr2);
}
```

A. 5

B. The address of ptr1

C. The address of a

D. The value of ptr1

21. Given the following code, what will this program print?

```
int main() {
    int a = 5;
    int *ptr1 = &a;
    int **ptr2 = &ptr1;
    printf("%x", ptr2);
}
```

A. 5

B. The address of ptr1

C. The address of a

D. The value of ptr1

22. Dynamic Memory Allocation uses which memory block?

A. code
B. data
C. stack
D. heap

22. Dynamic Memory Allocation uses which memory block?

A. code
B. data
C. stack
D. heap CORRECT

23. What will the following program print?

```
#include <stdio.h>
struct date {
    int day;
    int month;
    int year;
};
int main(void) {
    struct date d;
    printf("%u\n", sizeof(d));
    return 0;
}
```

A. 12 B. 8 C. 4 D. compilation error

23. What will the following program print?

```
#include <stdio.h>
struct date {
    int day;
    int month;
    int year;
};
int main(void) {
    struct date d;
    printf("%u\n", sizeof(d));
    return 0;
}
```

A. 12 B. 8 C. 4 D. compilation error

24. Which is the safer function?

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

24. Which is the safer function?

A. `gets()`

B. `fgets()` CORRECT

C. they are equally safe

D. they are equally unsafe

25. What will the following program print?

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

A. 126 CORRECT

B. 125

C. 100

D. error

25. What will the following program print?

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

A. 126 CORRECT

B. 125

C. 100

D. error

Thank you for coming!

Please write your BlazerID on the whiteboard before you leave.