Cpro (Monsoon '24) Quiz 1: Question and Solution

Instructors: Abhishek Deshpande, Sandeep Nagar, Girish Varma • Course Code: CS0.101 Duration: 45 mins

1 Short answer questions

1. Assuming 8 bits, express 33 in $2's$ complement notation. (5 marks)

   Solutions: First, write the positive number 33 in binary. 33 in binary (base 2) is 00100001. Science 33 is a positive number and its 2's complement will be the same as its binary representation.

2. What is the space taken by each of the following data types: char, double, int, float, short? (5 marks)

   Solution: char: 1 byte, double: 8 bytes, int: 4 bytes, float: 4 bytes, short: 2 bytes.

3. What is the output of the following program? Answer with justification. (5 marks)

```
1        int i = 2, j = 2, k = 2;
2        i += j += k;
3        printf("%d %d %d", i, j, k);
```

   Answer: output: 6 4 2
   Justification:

   - k remains 2 as it was not modified.
   - j is updated to 4 because $j+ = k$ where k was 2.
   - i is updated to 6 because $i+ = j$ after j was updated to 4.

4. What is the output of the following program? Answer with justification. (5 marks)

```
1    int n=0;
2    if (n >= 1 <= 10)
3        printf("n is between 1 and 10\n");
```

   Answer: n is between 1 and 10
   Justification: inside if condition, first it will check either $n \geq 1$ (which is false, 0 in C) then $0 \leq 10$. The condition does not check if n is between 1 and 10; it is a miswritten condition that still results in true due to how the operators are processed.

2 Multiple choice questions

In all questions, you can assume that the code is inside the body of a main function. Put a tick mark on the correct answer only.

Question 1: (3 marks)

What is the output of the following program?

```
1     int num = 2;
2     printf("%d", (num << 1) + (num >> 1));
```

(A) 2

(B) 1

(C) 4

(D) 5

Answer: D, output 5
Solution: Lets break down the expression:

- $num << 1$: This shifts the binary representation of num (which is 10 in binary) one bit to the left. This results in 100 in binary, which is equivalent to 4 in decimal.
- $num >> 1$: This shifts the binary representation of num one bit to the right. This results in 01 in binary, which is equivalent to 1 in decimal.
- $(num << 1) + (num >> 1)$: This adds the results of the two shift operations, 4 and 1, together. This gives us 5.

Therefore, the output of the code will be 5.

Question 2: (3 marks)

What is the output of the following program?

```
1     int i = 0, j = 0;
2     for (i = 0; i < 100; i++)  {
3         for (j = 0; j < 1;) {
4             break;
5         }
6         printf("CProgramming \n");
7     }
```

(A) CProgramming is printed 1 time.

(B) Nothing is printed.

(C) CProgramming is printed 100 times.

(D) Error

Answer: C, output: CProgramming ... (printed 100 times).
Solution:

- The inner loop doesn't affect the output directly. It's entered and immediately exited for each iteration of the outer loop.
- The break in the inner loop only exits that inner loop, not the outer one.
- The printf statement is executed once for each iteration of the outer loop.
- The variable j is reset to 0 at the start of each outer loop iteration, but it's not used meaningfully in this code.

This code structure is unusual and not typically how nested loops would be used efficiently. The inner loop and the j variable don't serve a practical purpose in this example.

Question 3: (3 marks)

What is the output of the following program?

```
1     int a=2, b=3;
2     printf("%d %d", a/b, b/a);
```

(A) 0 1

(B) 0.666666 1.5

(C) 0.66 1.500000

(D) None of these

Answer: A, output: 0 1
Solution:

- Integer division in C always results in an integer.
- The result is truncated, not rounded. This means any fractional part is simply discarded.
- If you need precise division with fractions, you should use floating-point numbers (float or double) instead of integers.
- The order of operands matters in integer division when the numbers aren't evenly divisible.

Question 4: (3 marks)
What is the output of the following program?

```
int i, j, count;
count=0;
for(i=0; i<5; i++);
{
    count++;
}
printf("%d",count);
```

(A) 1

(B) 5

(C) 0

(D) None of these

Answer: A, output: 1
Solution:

- The semicolon after the for loop is the crucial detail here. It creates an empty loop body.
- The block with count++ is not repeated; it executes just once after the loop.
- This is a common mistake in C programming, where an unintended semicolon can significantly change the program's behaviour.

Question 5: (3 marks)
What is the output of the following program?

```
int i = 0, j = 0;
while (i<5 & j<10)  {
    i++;
    j++;
}
printf("%d %d", i, j);
```

(A) 5 5

(B) 5 10

(C) Error

(D) 10 10

Answer: A, output: 5 5
Solution:

- While loop stops when i reaches 5, regardless of j's value.
- Both i and j are incremented the same number of times, so they end up with the same value.
- The use of & instead of && doesn't affect the outcome in this case, but it's generally not recommended. && is the proper logical AND operator in C.

Question 6: (3 marks)
What is the output of the following program?

```
int i = 3;
printf("%d \n ", i++);
printf("%d \n ", i++);
```

(A) 4 4

(B) 3 4

(C) 3 3

(D) 4 5

Answer: B, output: 3 4)
Solution: The postfix increment operator (i++) uses the current value of i in the expression, and then increments it. If we had used prefix increment (++i) instead, the output would have been different (4 and 5). After these operations, the final value of i is 5, although this isn't printed.
This code demonstrates the difference between the value of a variable and the side effects of operators. The postfix increment ensures that the "old" value is used in the expression before the increment takes place.

Question 7: (3 marks)
What is the output of the following program?

```
int a = 2;
switch(a) {
    case 1: printf("1 ");
    case 2: printf("2 ");
    case 3: printf("3 ");
    default: printf("None");
}
```

(A) 2.

(B) None.

(C) 1 2 3 None.

(D) None of these.

Answer: D, output: 2 3 None
Solution:

- a is initialized to 2.
- The switch statement checks the value of a.

- Since a is 2, execution starts at case 2:.
- There are no break statements after each case, so once a matching case is found, all subsequent cases are executed until the end of the switch block or until a break is encountered.
- With break statements, only "2 " would be printed for $a = 2$.

Question 8: (3 marks)
What is the output of the following program?

```
int x = printf("Hello!");
printf("%d", x);
```

(A) 5

(B) Error

(C) 1

(D) None of these

Answer: D, output: Hello!6
Solution: This code demonstrates an interesting feature of printf: it returns the number of characters it prints. This can be useful for error checking or, in this case, to determine the length of the output string.

Question 9: (3 marks)

What is the value of x, y, z after the following program (assuming the initial values are $x = 0, y = 0, z = 1$?

```
if (x)
    if (y)
        if (z)
            z = 3;
        else
            z = 2;
```

(A) 0 0 3

(B) 0 0 2

(C) 0 0 1

(D) Error

Answer: C, output: 0 0 1
Solution:

- First, it checks if x is true (non-zero in C, here $x = 0$ means false) and will not execute the nested if.
- If x is true, it checks if y is true (here $y = 0$ means false).
- If both x and y are true, it checks if z is true.
- If x, y, and z are all true, it sets $z = 3$.
- If x and y are true, but z is false, it sets $z = 2$.

Question 10: (3 marks)
What is the output of the following program?

```
double x = 0;
for (x = 0.0; x < 5.0; x++);
    printf("%lf \n ", x);
```

(A) 0.000000 1.000000 2.000000 3.000000 4.000000

(B) 4.000000

(C) 5.000000

(D) Error

Answer: C, output: 5.000000
Solution: The program prints 5.000000 because the loop increments x from 0.0 to 5.0, and when x reaches 5.0, the loop condition ($x < 5.0$) fails, leaving x with the value 5.0. Since ';' after the for loop, 'printf' will execute only once. x value is then printed.