CPro-M25 Quiz 1 Rubrik

Q1 - 5 Marks

Explanation - 56 in signed 8 bit Binary = `00111000`, taking two's complement, first flip all the bits to get `11000111` and add 1 to get `11001000`

[2] for writing `56 = 00111000`

[5] for all correct bits - `11001000`

[0] for anything else

Q2 - 3 Marks

```
printf("%d", ((x ^ y) < 0));
```

Explanation - Since sign is decided by the MSB, if both the integers are of same sign (same MSB), MSB will be 0 after xor, hence > 0 ⇒ outputs 0. If the two integers are of different sign (different MSB), MSB will be 1 after xor, hence < 0 ⇒ outputs 1

[1] for `computes xor of the two values and compares it with 0, if ≥ 0, prints 0, if < 0, prints 1`

[1] If assumed the operator as subtraction and written `outputs 0 if x ≥ y, outputs 1 if x < y`

[3] for `Outputs 0 if both the values are of same sign, outputs 1 if different sign`

[0] for anything else

Q3 - 3 Marks

```
1. for (i = 0; i < 10 ; i++)
2. for ( ; i < 10 ; i++)
3. for (i = 0; ; i++)
4. for (i = 0; i < 10 ; )
5. for ( ; ; )
```

Explanation - All for loops are valid, even though some fields are empty, the syntax is fine since there are semicolons.

[1] for ticking `option 1`

[+0.5] per option ticked from `options 2-5`

Q4 - 3 Marks

```
int i = 1, j;
for ( ; ; ) {
    if (i)          j = --i;
    if (j < 10)     printf("CproQuiz", j++);
    else            break;
}
return 0;
```

Explanation - First iteration j is set to 0, and then on, everytime `CproQuiz` is printed and j is incremented until it hits 10 when it breaks.

[1] for `CproQuiz`

[3] for `CproQuizCproQuizCproQuizCproQuizCproQuizCproQuizCproQuizCproQuizCproQuizCproQuiz` i.e CproQuiz repeated `10` times. No space or new line between each CproQuiz string.

[-1] if newline is omitted

[0] for anything else

Q5 - 3 Marks

```
int i, count;
count=0;

for(i=0; i<2025; i++); {
    count++;
}

printf("%d", count);
```

Explanation - Notice the semicolon after the for loop before the { braces. The for loop won't execute anything. After for loop, count will be incremented once.

[1] for `2025`

[3] for `1`

[0] for anything else

Q6 - 3 Marks

```
int p = 1;
while (_____) {
    int last = arr[0];
    for (int i = 0; _____; i++) {
        arr[i] = arr[i + 1];
    }
    _____
    p++;
}
```

Explanation - p starts from 1 and since we have to do cyclic shift d times, `p ≤ d` is the condition in while, which is the first fill in the blank. For the condition inside for loop, variable `i` shouldn't be `n-1` , since then `arr[i+1]` will result in RTE. Hence `i < n-1` , i.e only go till `i = n-2` , which is the second fill in the blank. To do a cyclic shift, we need to update the last element with the 0th element, hence `arr[n-1] = last` which would be the answer for the third.

[1] `p ≤ d`

[+1] `i < n-1`

[+1] `arr[n-1] = last;`

[0] for anything else

Binary marking for each fill in the blank. All `< , <=` signs shoul be correct. `;` should be included in the last answer.

Q7 - 3 Marks

```
char x = 6;
char y = ~ x << 2;

printf("%d\n", y);
```

Explanation - 6 in binary is `00000110` and since ~ has higher precedence than << (given), ~x is computed first which is `11111001` . Shifting all bits 2 positions to the left and adding 0s, we get `11100100`. Taking 2's complement to convert to decimal, we get `00011100` which is 28. Adding negative sign, since MSB is 1, we get `-28`

[1] for `~x = ~6 = ~00000110 = 11111001`

[+1] for `~x << 2 = 11100100`

[+1] for converting `11100100` to decimal which is `-28`

[3] for writing `-28`

[0] for anything else

Q8 - 3 Marks

```
signed char a = 100, b = 30;
signed char c = a + b;

printf("a+b = %d\n", c);
```

Explanation - When storing 130 i.e 100+30 in a signed char, it is stored with 1 in the MSB. C always stores signed datatypes in 2’s complement form. Converting `130 = 10000010` from 2’s complement form to decimal, the value becomes `-126` when we convert it to decimal.

[1] for `100+30 = 10000010` in 8-bit binary

[+1] for taking two’s complement `01111110`

[+1] for converting to decimal and adding - sign to get `-126`

[3] for `-126` . No extra characters need to be checked.

[0] for anything else

Q9 - 3 Marks

```
int x = 0, y = 5;

if (x++ && ++y)
    printf("Branch 1\n");

printf("x = %d, y = %d\n", x, y);

if (++x || y++)
    printf("Branch 2\n");
printf("x = %d, y = %d\n", x, y);
```

Explanation - Since it is post increment, x++ returns 0, and hence Branch1 is not printed. ++y is not executed because of short circuiting, but x++ updates x to 1. After this, x is 1 and y remains same i.e 5. After this, in the second if condition, ++x returns 2, hence `Branch 2` is outputted, but short circuit happens again, hence y++ is not executed again. x is incremented to 2 and y remains 5 which is the final output.

[+1] for `x = 1, y = 5`

[+1] for writing `Branch 2`

[+1] for `x = 2, y = 5`

[-1] if `Branch 1` is written

[0] if total marks for this ques < 0return 0;

Q10 - 3 Marks

```
int a = 3, b = 7, c = 2;
if (a++ > 3) {
    if (--b < 7) {
        printf("P5: X\n");
    } else {
        printf("P5: Y\n");
    }
} else {
    if (c++ == 2) {
        printf("P5: Z\n");
    } else {
        printf("P5: W\n");
    }
}
```

Explanation - First condition evaluates to False since it is post increment operation, and a = 3 is not > 3. In the second else condition, `c++ == 2` evaluates to True since it is post increment again i.e c++ returns 2 and 2 == 2, Hence, it prints `P5: Z`

[3] for `P5: Z`

[0] for any other option

Q11 - 3 Marks

```
int i, j;
for (i = 1; i <= 3; i++) {
   for (j = i; j <= 3; j++) {
      printf("%d ", i + j);
   }
   printf("\n");
}
```

Explanation - `i` goes from 1 to 3, while `j` goes from `i to 3`. For `i=1`, j goes from `1 to 3` and i+j is printed as `2 3 4` for each j. For `i=2`, j goes from `2 to 3` and `4 5` is printed on a new line and for `i=3`, j = 3, hence i+j = `6` is printed on a new line.

```
2 3 4
4 5
6
```

[+1] for each line of correct output. Note the spacing and new lines.

[0] for wrong formatting or wrong output

[-1] if new line is omitted

Q12 - 3 Marks

```
int a[] = {4, 3 , 5 , 2, 1};
int i = a[0];
while (i != 1) {
    printf("%d ", i);
    i = a[i - 1];
}
```

Explanation - i starts with `a[0] = 4` which is printed first. After this, `i = a[i-1] = a[4-1] = a[3] = 2` and hence 2 is printed next. After this, `i = a[i-1] = a[2-1] = a[1] = 3` and hence 3 is printed next. After this, `i = a[i-1] = a[3-1] = a[2] = 5` and hence 5 is printed next. Now, `i = a[i-1] = a[5-1] = a[4] = 1` and it doesn't print 1, it breaks out of the for loop.

[1] if `4` is written as the first output

[3] for all correct outputs - `4 2 3 5`

[-1] if `1` is printed as the last integer.

[-1] if newline is omitted

[0] for anything else or marks in this question < 0