Problem 1:
Convert the decimal integer 90 to 2-digit two's complement hexadecimal. (20 points)

ANSWER: 0x5a

\[ 90 = 5 \times 16 + 10 = 0x5a \]

Problem 2:
Convert the 8-bit two's complement binary integer 1001 0111 to decimal. (20 points)

ANSWER: -105

\[ 1001\ 0111 = -128 + 16 + 4 + 2 + 1 = -105 \]

Problem 3:
You're given two 4-digit, 2's complement hexadecimal numbers \( X = 0xc232 \) and \( Y = 0xf402 \). Compute \( X - Y \). Remember to indicate overflow if it occurs. Show all intermediate steps clearly. (30 points)

ANSWER: 0xce30 (no overflow)

\[-Y = \text{ffff} - \text{f402} + 1 = 0bfe\]

\[ X - Y = x + (-Y) = 0xc232 + 0x0bfe = 0xce30 \]
4) Circle a language: C++ or Java

Write a function/method in that language with the following prototype/header:

```c
void revArr(int arr[])
```

arr[] is an array of non-negative integers. It may be of any length; the end of the array is marked by the sentinel value -1. (For Java programmers, arr[] is allocated arr.length elements, but some of the elements are unused. The negative sentinel value marks the end of the valid elements in the array.)

revArr() reverses the order of the elements of the array arr[]. Write efficient code; obviously inefficient code will be penalized. **You are not allowed to call any library function that will reverse the array for you.**

For example:

```c
int x[] = {2, 17, 10, 13, -1, 4}; // x[] has 4 elements, 2, 17, 10, 13

revArr(x); // reverse order of x[], up to the sentinel value
```


**ANSWER:**

```c
void revArr(int arr[])
{
    // your code continues here...

    int i, count = 0;
    while (arr[count] != -1)
        count++;
    for (i=0; i<count / 2; i++) {
        temp = arr[i];
        arr[i] = arr[count - 1 - i];
        arr[count - 1 - i] = temp;
    }
}
```