Solution 1

```
#include <stdio.h>

unsigned int insertBits(unsigned int X, unsigned int N, unsigned int P) {

unsigned int mask = ~(0xFFFFFFFF << P);

        X &= mask;

        X |= (N << P);

return X;

}

int main() {

    unsigned int X = 10;

    unsigned int N = 5;

    unsigned int P = 6;

    unsigned int result = insertBits(X, N, P);

        printf("Result: %08X\n", result);

    X = 10;

    N = 5;

    P = 2;

result = insertBits(X, N, P);

    printf("Result: %08X\n", result);

    return 0;

}
```

Solution 2

```
#include <stdio.h>

#include <string.h>

void generic_swap(void* ptr1, void* ptr2, size_t size)

{

    char temp[size];

    memcpy(temp, ptr1, size);

    memcpy(ptr1, ptr2, size);

    memcpy(ptr2, temp, size);

}

typedef struct Student

{

    char a[10];

    int b;

    double c;

} Student;

int main()

{

    int i1 = 10, i2 = 20;

    float f1 = 1.6, f2 = 8.9;

    int a1[3] = {1, 2, 3}, a2[3] = {10, 20, 30};

    Student s1 = {"Adil", 42, 5.2}, s2 = {"Bilal", 9, 3};

    generic_swap(&i1, &i2, sizeof(int));

    printf("Swapped i1 and i2 -> %d and %d\n", i1, i2);

    generic_swap(&f1, &f2, sizeof(float));

    printf("Swapped f1 and f2 -> %.1f and %.1f\n", f1, f2);

    generic_swap(&a1, &a2, sizeof(a1));

    printf("Swapped a1 and a2 -> [%d, %d, %d] and [%d, %d, %d]\n", a1[0], a1[1], a1[2], a2[0], a2[1], a2[2]);
    generic_swap(&s1, &s2, sizeof(Student));
    printf("Swapped s1 and s2 -> {%s, %d, %.1f} and {%s, %d, %.1f}\n", s1.a, s1.b, s1.c, s2.a, s2.b, s2.c);
    return 0;
}
```

Solution 3

```
#include <stdio.h>
int main() {
    uint8_t decimalValue = 37;
    uint8_t bcdValue;
    bcdValue = DEC_TO_BCD(decimalValue);
    printf("Decimal: %d, BCD: 0x%X\n", decimalValue, bcdValue);
    bcdValue = decimalToBCD(decimalValue);
    printf("Decimal: %d, BCD: 0x%X\n", decimalValue, bcdValue);
    return 0;
}
```

Solution 4

a)

```
#include <stdio.h>

int main() {

    uint8_t num = 0x25;

    SET_BIT(num, 3);

    printf("After setting bit 3: %x\n", num);

    CLEAR_BIT(num, 1);

    printf("After clearing bit 1: %x\n", num);

    TOGGLE_BIT(num, 5);

    printf("After toggling bit 5: %x\n", num);

    return 0;

}
```

b)

```
#include <stdio.h>

void setBit(uint8_t *number, uint8_t bit) {

    *number |= (1 << bit);

}

void clearBit(uint8_t *number, uint8_t bit) {

    *number &= ~(1 << bit);

}

void toggleBit(uint8_t *number, uint8_t bit) {

    *number ^= (1 << bit);

}

int main() {

    uint8_t num = 0x25; // Example 8-bit number: 0010 0101
    setBit(&num, 3);

    printf("After setting bit 3: %x\n", num);

    clearBit(&num, 1);

    printf("After clearing bit 1: %x\n", num);

    toggleBit(&num, 5);

    printf("After toggling bit 5: %x\n", num);

    return 0;

}
```

Solution 5

```
#include <stdio.h>

#include <string.h>

typedef struct {

    char sensor_id[3];

    float temperature;

    int humidity;

    int light_intensity;

} SensorInfo;

void parseData(const char* data, SensorInfo* sensor) {

    char temp_str[10], hum_str[10], light_str[10];

    int temp_val, hum_val, light_val;

    sscanf(data, "%*[^S]S%s-T:%f-H:%d-L:%d", sensor->sensor_id, &sensor->temperature, &sensor->humidity,
&sensor->light_intensity);

}

int main() {

    char data[] = "S1-T:36.5-H:100-L:50";

    SensorInfo sensor;
    parseData(data, &sensor);
    printf("Sensor Info:\n");
    printf("_____________________\n");
    printf("Sensor ID: %s\n", sensor.sensor_id);
    printf("Temperature: %.1f C\n", sensor.temperature);
    printf("Humidity: %d\n", sensor.humidity);
    printf("Light Intensity: %d%%\n", sensor.light_intensity);
    return 0;
}
```

Solution 6

```
#include <stdint.h>
volatile uint8_t* INTCON_REG = (volatile uint8_t*)0x0B;
#define GIE_BIT     7
#define INTF0IF_BIT 1
void set_bit(uint8_t* reg, uint8_t bit_pos, uint8_t value) {
    if (value) {
        *reg |= (1 << bit_pos);
    } else {
        *reg &= ~(1 << bit_pos);
    }
}
void set_byte(uint8_t* reg, uint8_t value) {
    *reg = value;
}
int main() {
    set_bit(INTCON_REG, GIE_BIT, 1);
    set_bit(INTCON_REG, INTF0IF_BIT, 1);
```

```
 indicate an external interrupt on INT0
    set_byte(INTCON_REG, 0x82);
    return 0;
}
```

Solution 7

```
#include <stdio.h>
#include <string.h>
void removeDuplicateChar(char *s, char c) {
    int len = strlen(s);
    int currentIndex = 0;
    for (int i = 0; i < len; i++) {
        if (s[i] != c || (i > 0 && s[i - 1] != c)) {
            s[currentIndex] = s[i];
            currentIndex++;
        }
    }
    s[currentIndex] = '\0';
}
int main() {
    char inputString[100];
    char charToRemove;

    printf("Enter the input string: ");
    fgets(inputString, sizeof(inputString), stdin);

    printf("Enter the character to remove: ");
    scanf("%c", &charToRemove);
    int len = strlen(inputString);

    if (inputString[len - 1] == '\n')

        inputString[len - 1] = '\0';

    removeDuplicateChar(inputString, charToRemove);

    printf("Result: %s\n", inputString);

    return 0;

}
```