Question 1: Function and Pointer (10 pts)
The following function is intended to calculate the sum and product of two integers in a single function call:

```c
#include "stdio.h"

// calculate the sum and product of two integers
void sum_and_product (int sum, int product, int x)
{
    sum = x + (x+1);
    product = x * (x+1);
}

// example of calling sum_and_product()
void call_example()
{
    int sum, product;

    sum_and_product (sum, product, myarray[0]);

    printf("sum = %d, product = %d", sum, product);
    // Desired output of printf: sum = 30, product = 200
}
a. [5 pts] The function does not work as intended. Explain Why.

1) sum and product are passed by value to the function, thus the function sum_and_product cannot modify them. They need to be passed by address.

2) my_array should pass the address of the array, not the first element of the array

3) If x is the address of my_array, then the proper values can be access by using array access or pointer access.

b. [5 pts] Revise the code of: sum_and_product and call_example so that they behave as desired. Test your code, and then cut and paste it in the space below. Highlight your changes in bold type.

Note 1: You may not modify the number of arguments to the functions.
Note 2: You may not modify how many lines of code are in the functions, you may only modify the existing lines of C code.

```c
// calculate the sum and product of two integers
void sum_and_product (int *sum, int *product, int x[]) {
    *sum = *x + *(x+1); // OR x[0]+ x[1];
    *product = *x * *(x+1); // OR x[0] * x[1];
}

// example of calling sum_and_product()
void call_example() {
    int sum, product;

    sum_and_product (&sum, &product, myarray[0]);

    printf("sum = %d, product = %d", sum, product); // Desired output of printf: sum = 30, product = 200
}
```
Question 2: Struct and Union (10 pts)
What is the size of the following data structures in bytes, assuming we’re using the Atmel compiler?

a) 2 pts.

```c
struct point3D {
    signed long x; // 4 bytes
    unsigned int y; // 2 bytes
    char z; // 1 byte
};
```

`4 + 2 + 1 = 7 bytes`

b) 2 pts.

```c
union val {
    char cval; // 1 byte
    char str; // 1 byte
    int ival[8]; // 16 bytes
    float fval; // 4 bytes
    double dval; // 4 bytes
};
```

`16 bytes`

c) 3 pts.

```c
struct compound {
    char *mystring; // 2 bytes
    long y; // 4 bytes
    union {
        char *c; // 2 bytes
        int *i; // 2 bytes
        float *f; // 2 bytes
    } u;
};
```

`2 + 4 + 2 = 8 bytes`

d) 3 pts.

```c
struct more_compound {
    char *name; // 2 bytes
    int age; // 2 bytes
    long pay; // 4 bytes
    long *height; // 2 bytes

    union {
        char short_id; // 1 byte
        int normal_id; // 2 bytes
        char my_id; // 1 byte
    } id;

    union {
        char *text_data; // 2 bytes
        int *numeric_data; // 2 bytes
    }
};
```
long l_numeric_data; // 4 bytes 
} data;
};
2+2+4+2+4 = 16 bytes

**Question 3: Pointers (10 pt)**
What is the value of a, b, c_ptr at the end of each program. Give N/A if a value is undefined.

Assume these programs would be run on the ATMega128.

Assume the following memory locations for the variables:
- Location of a is 0xFF00
- Location of b is 0xFE00
- Location of c_ptr is 0xFD00

**a. 3pts**
void main()
{
  char a = 5;
  char b = 10;
  char *c_ptr;

  c_ptr = &a; // c_ptr = 0xFF00
  *c_ptr = 12; // Go to 0xFF00 set to 12, i.e. sets a = 12
  *c_ptr = b; // Go to 0xFF00 set to b, i.e. sets a = b == 10.
  c_ptr = &b; // c_ptr = 0xFE00
  *c_ptr = a; // Go to 0xFE00 set to a, i.e. sets b = a == 10;
}

a is **10**  b is **10**  c_ptr is **0xFE00**.

**b. 3pts**
void main()
{
  char a = 5;
  char b = 10;
  char *c_ptr = 0;

  c_ptr = &a; // c_ptr = 0xFF00
  c_ptr = &b; // c_ptr = 0xFE00
  (*c_ptr)++; // Go to 0xFE00 and increase by 1, i.e. b++ == 11
  c_ptr++; // c_ptr = c_ptr + (1 * 1) = 0xFE01
}

a is **5**  b is **11**  c_ptr is **0xFE01**.
void main()
{
    int a = 5;
    int b = 10;
    int *c_ptr = 0;

    c_ptr = &b;     // c_ptr = 0xFE00
    a = *c_ptr + b; // Go to 0xFE00 and increase by b,
        // i.e. a = b + b == 20
    (*c_ptr)++;    // Go to 0xFE00 and increase by 1, i.e. b++ == 11
    c_ptr++;    // c_ptr = c_ptr + (1 * 2) = 0xFE02
}

a is 20    b is 11    c_ptr is 0xFE02.

**Question 4: C-Strings and pointers (10 pts)**

a. [5] The following function finds the length of a C-string (not counting the NULL byte). Complete the function using array access, i.e. str[i], but no pointer access.

   **Note:** For both part a) and part b) you may **not** use any library function calls (e.g. you cannot use strlen, etc.)

```c
int my_strlen(char str1[])
{
    int length = 0; // Store length of C-string str1
    int i = 0;

    while(str1[i] != 0) //Test for NULL byte (i.e. end of C-string)
    {
        length++; // increase length by 1
        i++ // move to next byte in C-string
    }

    return length;
}
```
b. [5] Rewrite the above function, this time using pointer access, i.e. *str, but no array access.

Note: In general, a function’s parameter declarations “char str[]” and “char *str” are equivalent (i.e. they are interchangeable)

```c
int my_strlen(char *str)
{
    int length = 0;  // Store length of C-string str
    int i = 0;

    while(*str+i != 0) //Test for NULL (i.e. end of C-string)
    {
        length++;  // increase length by 1
        i++;     // move to next byte in C-string
    }

    return length;
}
```
**Question 5: Small Program (10 pts)**

Complete the following function that calculates the cost of a grocery list. Each element in the array `X[]` represents an item in the list. If an element has an odd value, then the cost of the item is 5. If the element has an even value, then the cost is 10.

Hint: using the `%` (i.e. modulus) operator may be useful

```c
// Calculate the cost of a grocery list. X is the array of
// items, size is the array size. Return the total cost of the
// grocery list.
int calculate_cost(int X[], int size)
{
    int i, cost = 0;

    for (i = 0; i < size; i++)
    {
        switch (X[i] % 2)
        {
            case 0:
                cost += 10;
                break;
            case 1:
                cost += 5;
                break;
        }
    }
    return cost;
}
```