1. What can you learn from the following program? (From the perspective of variable content). Write the corresponding C code next to the assembly instruction

```
int main(void)
{
    int a = 1;
    int *p = &a;
    int b = 2;
    char *str = "BDCOM";
    int array[3] = {1, 2, 3};
    int *q = array;

    return 0;
}
/*
LC0:
    .string    "BDCOM"
    .text

main:
    pushl    %ebp
    movl     %esp, %ebp
    subl     $32, %esp
    movl     $1, -4(%ebp)
    leal     -4(%ebp), %eax
    movl     %eax, -8(%ebp)
    movl     $2, -12(%ebp)
    movl     $.LC0, -16(%ebp)
    movl     $1, -32(%ebp)
    movl     $2, -28(%ebp)
    movl     $3, -24(%ebp)

    leal     -32(%ebp), %eax
    movl     %eax, -20(%ebp)
    movl     $0, %eax
    leave
    ret
*/
```

a. I have learned from the program from the perspective of variable content are given below:

| Variable | Type | Content | Memory Location |
|---|---|---|---|
| a | int | 1 | 0x100 |
| p | int* ( Pointer ) | Address of a (0x100) | 0x104 (stores address of a) |
| b | int | 2 | 0x108 |
| str | char* ( Pointer ) | Address of string “BDCOM” | 0x200(points to character ‘B’ ) |
| array | int[3] (array) | {1, 2, 3} | 0x300, 0x304, 0x308 |
| q | int* (Pointer) | Address of array[0] (0x300) | 0x312 |

b. corresponding C code next to the assembly instruction

```
LC0:
    .string "BDCOM"       ; Stores the string "BDCOM" in the read-only section
    .text                 ; Start of text (code) section
main:
    pushl %ebp            ; Save old base pointer
    movl %esp, %ebp       ; Set new base pointer
    subl $32, %esp        ; Allocate 32 bytes of stack space for local variable

    movl $1, -4(%ebp)     ; a = 1;
    leal -4(%ebp), %eax   ; Load address of a into EAX
    movl %eax, -8(%ebp)   ; int *p = &a;

    movl $2, -12(%ebp)    ; b = 2
    movl $.LC0, -16(%ebp) ; char *str = "BDCOM"; (string in read-only memory)

    movl $1, -32(%ebp)    ; array[0] = 1;
    movl $2, -28(%ebp)    ; array[1] = 2;
    movl $3, -24(%ebp)    ; array[2] = 3;

    leal -32(%ebp), %eax  ; Load address of array[0] into EAX
    movl %eax, -20(%ebp)  ; int *q = array;

    movl $0, %eax         ; return 0
    leave                 ; Restore stack
    ret                   ; Return to caller
```

c.

2. There is a program, p,*p and &p, Answer following questions

```c
int main(void)
{
    int a = 1;
    int *p = &a;

    return 0;
}
```

a. Shortly compare between a, &a, p, *p, &p?

| **Expression** | **Meaning** | **Type** | **Value** |
|---|---|---|---|
| a | Variable a itself | int | 1 |
| &a | Address of a | int* ( pointer to int) | Address of a (e.g. 0x1000 ) |
| p | Pointer storing &a | int* | Address of a (e.g. 0x1000 ) |
| *p | Dereferencing p | int | 1 ( a ) |
| &p | Address of pointer p | int** ( pointer to int* ) | Address of p ( e.g. 0x2000 ) |

b. Which can be used as the l-value, and which can not in *p,p and &p?

| **Expression** | **Can Be Used as L-Value?** |
|---|---|
| p | Yes |
| *p | Yes |
| &p | No |

c. Suppose the address of variable a is 0x1000, p is 0xFFC, Fill the table

| **Variable, Address** | **a (0x1000)** | **p ( 0xFFC )** |
|---|---|---|
| Raw Value | ( Unknown ) | ( Unknown ) |
| *p = 2 | 2 | 0x1000 |
| p = 1 | 2 | 1 |
| *p = 2 | Segmentation fault or undefined behavior | 1 |

3. What can you learn from the following program? (From the perspective of pointer data type)

```
int main(void)
{
    int a = 0x11223344;
    int b = 0x12345678;
    int *p = &b;
    char *q = &b;

    printf("%d %d\n", sizeof(p), sizeof(q));    // 4 4

    printf("%p %x\n", p, *p);                   // 0xbfb51508 12345678
    printf("%x\n", *(unsigned char*)p);         // 0xbfb51508 78
    p = (unsigned char*)p + 1;
    printf("%x\n", *(unsigned char*)p);
            // 0xbfb51509 56

    p = &b;
    p++;
    printf("%p %x\n", p, *p);    // 0xbfb5150c 11223344

    printf("%p %x\n", q, *q);    // 0xbfb51508 78
    q++;
    printf("%p %x\n", q, *q);    // 0xbfb51509 56
    q++;
    printf("%p %x\n", q, *q);    // 0xbfb5150a 34
    q++;
    printf("%p %x\n", q, *q);    // 0xbfb5150b 12
    q++;
    printf("%p %x\n", q, *q);    // 0xbfb5150c 44

    return 0;
}
```

a. I have learned from the program is given below:

    1. Pointer Size and Alignment : Pointer size is fixed for all type of pointer.
        a. 4 byte for 32 bits os
        b. 8 bytes for 64 bits os.

    2. Pointer Type Affects Dereferencing :
        a. int *p reads 4 bytes starting from the memory address.
        b. char *q reads 1 byte starting from the memory address.

    3. Pointer arithmetic depends on the data type:
        a. int *p++ moves 4 bytes forward.
        b. char *q++ moves 1 byte forward.

    4. Memory alignment follows the Little Endian format :
        a. 0x12345678 is stored in memory from lower to higher address as 78 56 34 12

    5. Casting to char * allows reading memory byte-by-byte, useful for working with binary data .

4. What is the wild pointer? Illustration
    a. A wild pointer is a pointer that has not been initialized and therefore contains a garbage (random) memory address. Accessing or modifying memory through a wild pointer leads to undefined behavior, which may cause crashes, corruption, or unpredictable results.
    b. Illustration:

```c
#include <stdio.h>

int main() {
    int *ptr;  // Wild pointer (not initialized)

    *ptr = 10; // Undefined behavior. It points to a random memory location
    printf("%d\n", *ptr);

    return 0;
}

```

    c. **Output:**
    **Segmentation Fault ( Run linux server )**

5. What is dangling pointer? Illustration
    a. A dangling pointer is a pointer that points to a memory location that has been freed, deleted, or gone out of scope. Dereferencing a dangling pointer leads to undefined behavior(garbage value), which can cause crashes, memory corruption, or security vulnerabilities.
    b. Illustration:

```c
#include <stdio.h>
#include <stdlib.h>

int main() {
    int *ptr = (int*)malloc(sizeof(int)); // Allocate memory
    *ptr = 42;

    free(ptr); // Free memory, but ptr still holds the old address (dangling)

    printf("\n%d\n", *ptr); // Undefined behavior (dangling pointer access)

    return 0;
}

```

    c. **output**: garbage value

```
-249537920
PS C:\Users\Admin\Documents\bdcom_coding_zone\pointer>
```