1. Consider the following code:

```
char *p = "abc";
printf("%s %s\n", p, p+1);
```

   What is printed? Explain briefly.

   Answer:

   abc bc

   p points the start of the string. Since p is a pointer to an element in an array, then (p+1) points to the next element in the array, i.e. 'b', and prints the string starting from there.

2. Given that the & operator returns the address of the variable, consider the following code:

```
#include <stdio.h>

int main()
   {
    int a[]={10, 20, 30, 40, 50};
    int *ptr;
    ptr=a;
    printf("%u %u %u", &a[0],ptr,&ptr);
    return 0;
}
```

   Assume that the starting memory location of array **a** is 2147478270, what is the output of the above code segment? Explain.

   Answer:

   2147478270 2147478270 2147478270

   The & (address of) operator normally returns the address of the operand. However, arrays are an exception. When applied to an array (which is an address), it has the same value as the array reference without the operator. This is not true of the equivalent pointers, which have an independent address.

3. Array / pointer operations – consider the following code segment:

```
#include <stdio.h>
#define MAX 10

int main() {
  int a[MAX];
```

```
    int b[MAX];
    int i;
    for(i=0; i<MAX; i++)
        a[i]=i;
    b=a;
    for(i=0; i<MAX; i++)
        printf("b[%d] = %d\n", i, b[i]);
    return 0;
}
```

Is the code valid? Will it cause a compiler error? If yes, where and why. If not, what would the output be?

Answer:

If it is compiled, there will be an error. Arrays in C are unusual in that variables **a** and **b** are not arrays themselves but permanent pointers to arrays. Thus, they point to blocks of memory that hold the arrays. They hold the addresses of the actual arrays, but since they are pointer/address constant, their addresses cannot be changed. The invalid statement is **b=a;** and will generate a compiler error.

4. Two dimensional arrays – consider the following:

```
#define MAX_ROWS 10
#define MAX_COLS 10
int A[MAX_ROWS][MAX_COLS];
```

When data is accessed in our matrix using the notation, **A[i][j]**, the location for this data is computed using:

&A[0][0] + MAX_COLS * i + j

The base address, starting address, of the 2D array can be referred to by: **&A[0][0]**, **A[0]**, **A** and also **&A[0]**. What do the following point to, in **A[i][j]** format? Explain your answer.

i.	&A[0][0] + 1	→	A[ ][ ]
ii.	A[0] + 1	→	A[ ][ ]
iii.	A + 1	→	A[ ][ ]
iv.	&A[0] + 1	→	A[ ][ ]

Answer:

i.	&A[0][0] + 1	→	A[0][1]	base location plus one element
ii.	A[0] + 1	→	A[0][1]	base (row) location plus one element in row
iii.	A + 1	→	A[1][0]	base location plus one row
iv.	&A[0] + 1	→	A[1][0]	base (row) location plus one row

5. What is the difference between the following two declarations?
    i. int (*ptr2arr)[10];
    ii. int *ptr2arr[10];

Answer:

iii. int(*ptr2arr)[10]; $\rightarrow$ ptr2arr is a pointer to an array of 10 integers

iv. int *ptr2arr[10]; $\rightarrow$ ptr2arr is an array of 10 pointers to type int.