Assignment - 8

Q.1 What is meant by "virtual memory address"?

--> virtual memory address is a pointer or marker for a memory space that an OS allows a process to use.

- The virtual address points to a process location in a primary storage that a process can in primary storage use independently of other processes
- with virtual addresses, memory management system is able to allocate huge amount of memory to a process.

2. Predict the output of below code snippet

```
#include <stdio.h>

int main()
{
    int arr[6] = {10, 20, 30};   // base address as 100
    int no = 2;
printf("%d", arr[0]);      // 10
printf("%d", arr[no]);     // 30
printf("%d", arr[3-1];     // 30
printf("%d", arr);         // 100
printf("%d", arr+1);       // 104
```

Assignment - 8

Q.1 What is meant by virtual memory address?

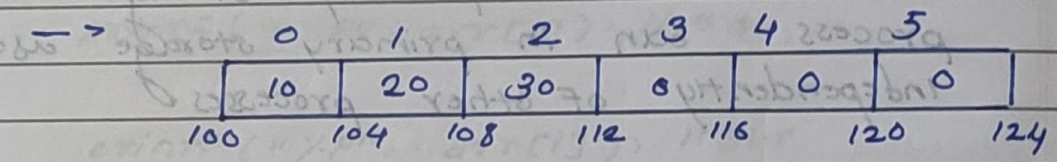
-> virtual memory address is a pointer or marker for a memory space that an OS allows a process to use.

- The virtual address points to a process location in a primary storage that a process can in primary storage use independently of other processes.
- With virtual addresses, memory management system is able to allocate huge amount of memory to a process.

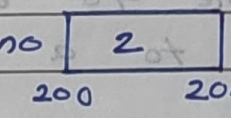
2. Predict the output of below code snippet

```
#include <stdio.h>

int main()
{
    int arr[6] = {10, 20, 30};   // base address as 100
    int no = 2;
printf("%d", arr[0]);    // 10
printf("%d", arr[no]);   // 30
printf("%d", arr[3-1];   // 30
printf("%d", arr);       // 100
printf("%d", arr+1);     // 104
```

```
printf ("%d", (&arr)+1); // 104
printf ("%d", arr+3); // 112
printf (%d (arr[3])); // 112
printf ("%d", arr[4]); // 0
printf ("%d", &(arr[5])); // 120
printf ("%d", 2[arr]); // 30

return 0;
}
```




```
int no = 2;
```



o/p - 10
30
30
100
104
104
112
112
0
120
30

Q.3 What is a use of & (address of) operator ?

→ • Address of operator is used by every variable to access their memory address.

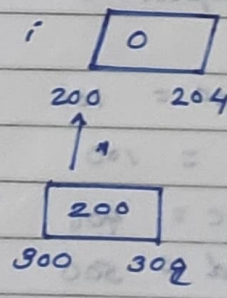
• we can we use address of operator to access a memory location of an variable.

• we can get a were a data is stored exactly in a memory.

• Pointer uses '&' operator (&) address of operator stores the address of any variable & we can access that variable with the help of address.

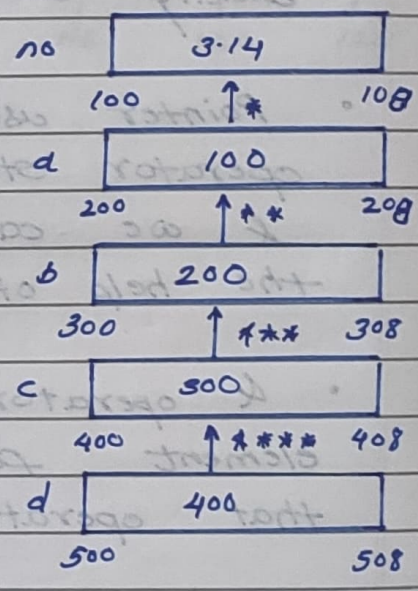
• & operator helps the user to access the element fastly that the direct calling that operatior.

e.g. - int i = 0;
int * ip = &i;



Q.4. output with diagramatic representation.

```
#include <stdio.h>
int main()
{
    double no = 3.14;        // consider address 100
    double *a = &no;         // address 200
    double **b = &a;         // address 300
    double ***c = &b;        // address 400
    double ****d = &c;       // address 500

    printf("%d", &no);
    printf("%d", a);
    printf("%d", &c);
    printf("%d", &d);
    printf("%d", d);
    printf("%d", **d);
    printf("%d", **c);
    printf("%d", *b);

    return 0;
}
```



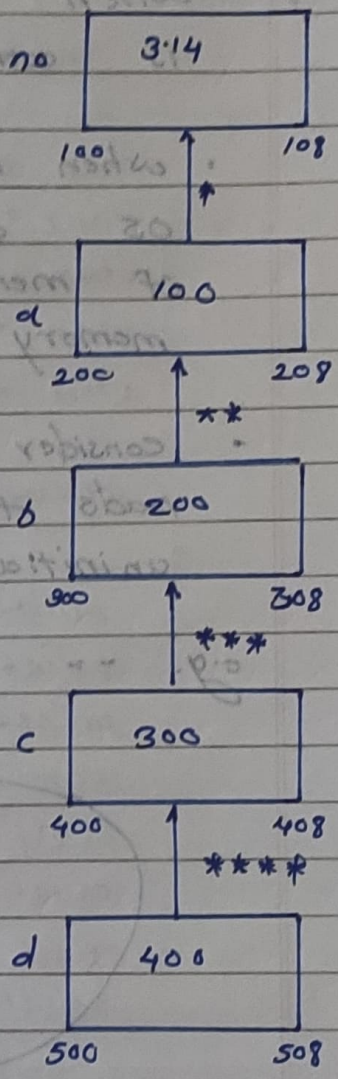
→ output =

&no = 100
a = 100
&c = 400
&d = 500
&a = 200

**d = 200
**c = 100
*b = 100
d = 400

Q5. Predict the output & draw diagrammatic representation.

```
#include <stdio.h>
int main()
{
    double no = 3.14;
    double *a = &no;
    double **b = &a;
    double ***c = &b;
    double ****d = &c;
    return 0;
}
```



→

sizeof(no) = 8 byte

sizeof(a) = 8

sizeof(b) = 8

sizeof(c) = 8

sizeof(d) = 8

sizeof(**d) = 8

sizeof(****d) = 8

sizeof(*a) = 8

sizeof(***c) = 8

sizeof(**c) = 8

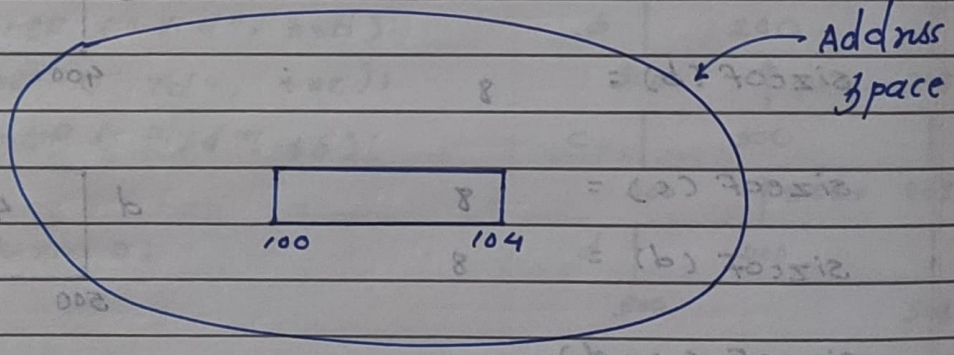
Q.6. What is meant by address space of process ?

→ • When we execute any program it is consider as a process.

• When any process gets executed the OS will allocate some amount of memory for a process and that memory is called as address space.

• Consider the below scenario which may leads to segmentation fault due to uninitialized pointer.

e.g. →



Q.7. what is the use of Nat instruction member of stack frame?

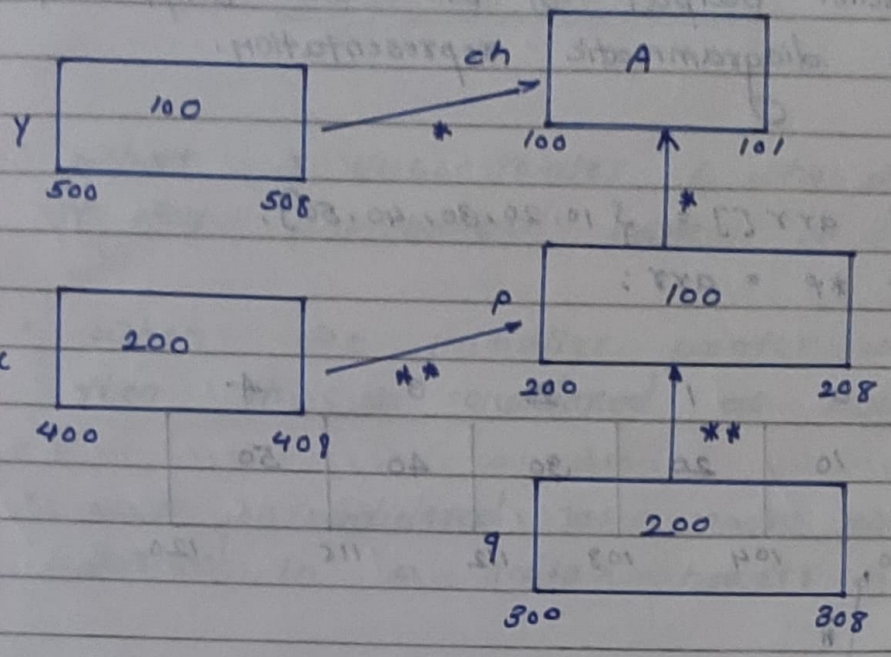
→

Q.8.

9. Predict the output of below code snippet & draw its diagrammatic representation.

```
#include <stdio.h>
int main()
{
    char ch = 'A';
    char *p = &ch;
    char **q = &p;
    char **x = &p;
    char *y = &ch;

    return 0;
}
    printf("%d", &ch);
    printf("%d", p);
    printf("%d", &p);
    printf("%d", &q);
    printf("%d", d);
    printf("%c", **x);
    printf("%c", **q);
    printf("%d", *q);
    return 0;
}
```

→



→ &ch = 100

p = 100

&p = 200

&q = 300

d = Garbage (Not declared)

** x = A

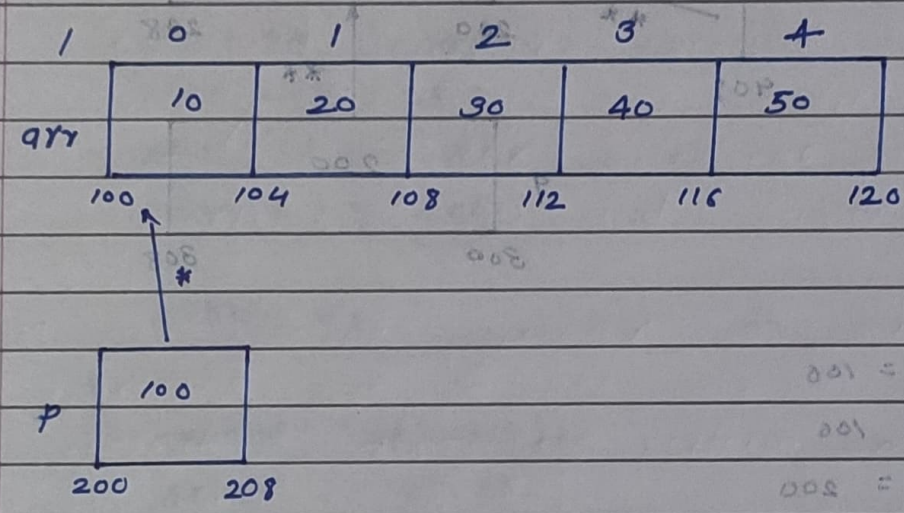
**q = A

*q = 100

Q. 10 predict output of below code snippet & draw its diagrammatic representation.

```
int arr[] = {10, 20, 30, 40, 50};
int *p = arr;
```

→



→

```
printf("%d", arr);    → 100
printf("%d", &arr);   → 100
printf("%d", p);      → 100
printf("%d", *p);     → 10
printf("%d", sizeof(arr));    → 120 byte
printf("%d", sizeof(arr[0])); → 4 byte
printf("%d", sizeof(p));      → 8 byte
printf("%d", sizeof(*p));     → 204 byte
```