Assignment No-8

Q1] What is mean by virtual address.

→ A virtual address is an address used by a program to access memory in a Computer System.

- It is part of the virtual memory system implemented by modern operating systems.
- Virtual addresses allow programs to use memory as if they have a large, contiguous block of addresses, even though the actual physical memory may be fragmented or limited.

Q2] What is use of & (address of) operator.

→ The & operator known as the address-of operator. It is used to obtain memory address of a variable.

This operator is crucial in working with pointers, as it allows you to assign the address of variable to a pointer.

Uses-

- Finding the Address of a variable - The & operator provides the memory address where the variable is stored.
- Pointer Initialization - It is used to assign the address of a variable to a pointer.

3] What is address space of Process.

→ It is a consider as memory allocated for the process when its gets loaded into the RAM.

Address Space of process

argc, argv		Command line arguments	
Info of fun?, local variable		Stack section	
	↓	Potential Gap	
Dynamically allocated variable, obj		Heap section	Data section
non-initialized global variable	{ Data }	BSS section	
Initialized global variable	{ Section }	No BSS section	
Static variable		Static Section	
Program binary instruction		Text Section	

Consider above sections are executable file file.exe loaded in RAM then it became process

- An each process is having address space & each have different space.

- **Command line Arguments**

when we pass parameters at the time of execution then it is considered as command line argument.

- **Stack**

This section of process contains info about function that we call from our program.

In form of stack frame.

- Section stack contains memory for local variabl that we use in our program.

- **Potential Gap**

This is empty section of process which can be used by either stack or heap section.

- This section reserved for dynamic memory allocation for a program.

Stack overflow - when stack section memory from

potential gap then OS allots stack overflow.
outoff Memory - when heap section uses full potential gap then OS allots outoff memory
• Data Section
This section contain memory for global gap ~~then OS allots outoff memory~~ memory of program. (external storage class variables)
- Data section logically divided into two parts BSS section., Non BSS section.
BSS Section
It is a part of Data section it considered as block starting with symbol.
It contains memory for non-initialized global variable.
non-BSS
consider as block starting with value.
Contains initialized global variable.
Static Section
It is not a separate section, it also consider as part of data section.
• Text Section
It contains compiled instructions of a programs which are in the binary format.
All function that we defined in program gets converted into binary. That instruction get stored in text section.

Q4] Predict the output of below code.

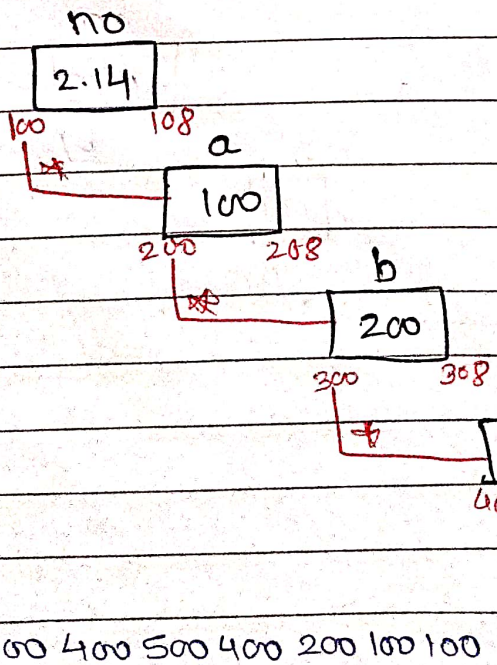
```
#include<stdio.h>
int main()
{
    int arr[6] = {10, 20, 30};
    int no = 2;
    printf("%d", arr[0]);
    printf("%d", arr[no]);
    printf("%d", arr[3-2]);
    printf("%d", arr);
    printf("%d", arr+1);
    printf("%d", &(arr)+1);
    printf("%d", arr+3);
    printf("%d", &(arr[3]));
    printf("%d", arr[4]);
    printf("%d", &(arr[5]));
    printf("%d", 2[arr]);
    return 0;
}
```

output-

```
10
30
20
6422276
6422280
6422300
6422288
0
6422296
30
```

Q5 Predict the output of below code & its diagrammatic representation.

```
#include<stdio.h>
int main()
{
    double no = 2.14
    double *a = &no;
    double **b = &a;
    double ***c = &b;
    double ****d = &c;

    printf("%d\n", &no);
    printf("%d\n", a);
    printf("%d\n", &c);
    printf("%d\n", &d);
    printf("%d\n", d);
    printf("%d\n", **d);
    printf("%d\n", **c);
    printf("%d\n", *b);
    return 0;
}
```

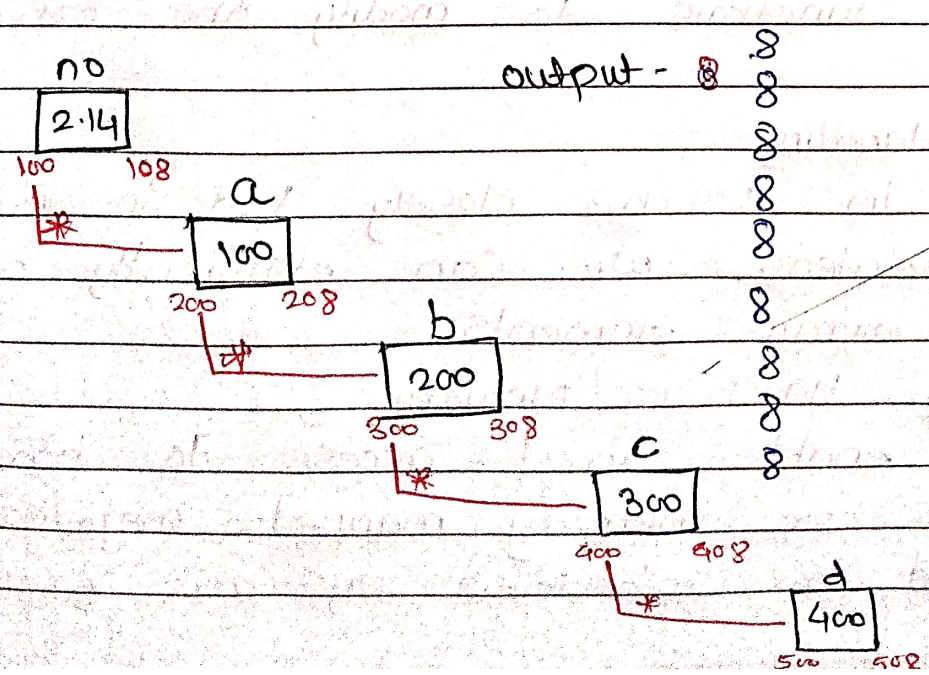


Output - &no = 100
*a = 100
b = 200
c = 300
d = 400
*d = 300
**d = 200
***d = 100
*c = 300
**c = 200
***c = 100

output- 100 100 400 500 400 200 100 100

Q6 Predict the output of below code & draw diagrammatic representation.

```
#include<stdio.h>
int main()
{
    double no = 2.14; // consider address of no as 100
    double *a = &no; // consider address of a as 200.
    double **b = &a; // consider address of b as 300.
    double ***c = &b; //
    double ****d = &c;

    printf("%d \n", sizeof(no));
    printf("%d\n", sizeof(a));
    printf("%d\n", sizeof(b));
    printf("%d\n", sizeof(c));
    printf("%d\n", sizeof(d));
    printf("%d\n", sizeof(**d));
    printf("%d\n", sizeof(**c));
    printf("%d\n", sizeof(****d));
    printf("%d\n", sizeof(***c));
return 0;
}
```



output - 8
8
8
8
8
8
8
8
8
8

Q7 What is pointer in c programming? why to use it.

→ Pointer is considered as derived data type in C, C++.

- Pointer is considered as variable which stores memory.
- Pointer is such a special variable which stores address of any thing.
- the power of pointer is to fetch the data whose address is stores in it.
- Pointer is a variable which holds address each address is of type unsigned long due to which size of every pointer is 8 byte.

• Why use Pointers in C

- Dynamic memory Allocation
Pointers are essential for allocating memory dynamically using functions like malloc() and calloc() in the heap.
- Efficient function Arguments (Pass by reference)
Pointers allow passing variables by reference, enabling functions to modify the original values.
- Array Handling
Aarrays in c are closely tied to pointers using pointers, we can efficiently navigate through array elements.
- Accessing Hardware/ Memory
Pointers enables direct access to hardware resources or memory mapped registers in ~~embedded~~ embedded systems programming

- Data structures
- Pointers are crucial for implementing dynamic data structures like linked lists, trees and graphs.
- Improved Performance.
Instead of copying large structures, pointers allow working on memory locations directly saving time and space.

Q8 What are different usage of pointer; explain with examples.

→ 1) Accessing variables using memory address
Pointers can store the address of a variable and access its value using dereferencing.
eg-

```
#include <stdio.h>
int main()
{
    int x = 10;
    int *ptr = &x;
    printf("Address of x: %d\n", &x);
    printf f;
    return 0;
}
```

2) Dynamic Memory Allocation
Pointers are used for allocating and deallocating memory dynamically at runtime.
eg-

```
#include<stdio.h>
int main()
{
```

```
int *arr = (int *)malloc(3 * sizeof(int));
arr[0] = 10;
arr[1] = 20;
arr[2] = 30;
printf("Dynamic Array Elements: %d %d %d\n",
        arr[0], arr[1], arr[2]);
return 0;
}
```

3) Array Traversal

Pointers can traverse arrays more efficiently by using pointer arithmatic.

eg-

```
#include <stdio.h>
int main()
{
  int arr[] = {10, 20, 30, 40};
  int *ptr = arr;
  for (int i = 0; i < 4; i++)
      { printf("arr[%d] = %d\n", i, *(ptr + i)); }
  return 0;
}
```

4) Passing variables by reference.

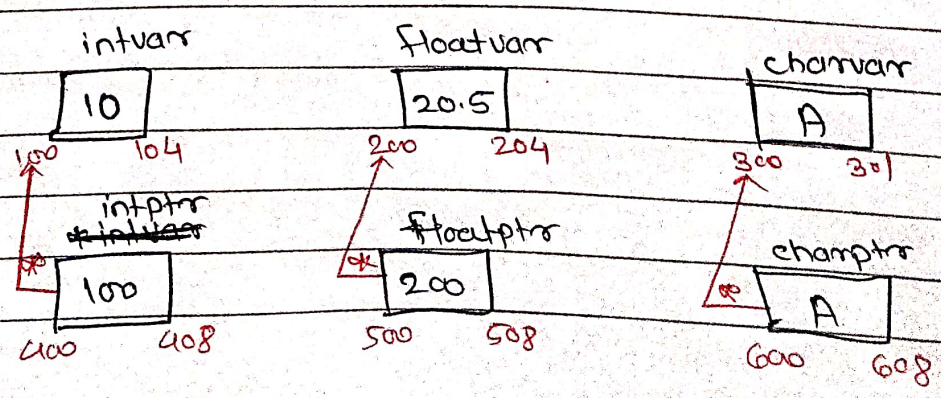
Pointers allow modifying the actual value of a variable passed to a function.

eg-

```
#include <stdio.h>
void modify (int *p){
{
```

Q9] Write a program to demonstrate the different types of pointer which are pointing to primitive data types.

```
#include <stdio.h>
int main()
{
    int intvar = 10;
    float floatvar = 20.5;
    char charvar = 'A';

    int *intptr = &intvar;
    float *floatptr = &floatvar;
    char *charptr = &charvar;

    printf("value of intvar : %d\n", *intptr);
    printf("address of intvar : %d\n", intptr);

    printf("value of floatvar : %d\n", *floatptr);
    printf("address of floatvar : %d\n", floatptr);

    printf("value of charvar : %d\n", *charvar);
    printf("address of charvar : %d\n", charvar);

return 0;
}
```



Q10] read the below statements and write readline statements for the same int no = 10;

a) int no = 10;
int *p = &no;

→

printf("value of no: %d\n", no);
printf("Address of no: %d\n", p);

b) char ch = 'A';
char *chptr = &ch;

→

printf("value of ch: %c\n", ch);
printf("Address of ch: %d\n", chptr);

c) double d = 10202020;
double *dbptr = &d;

→

printf("value of d: %d\n", *d);
printf("Address of d: %d\n", dbptr);