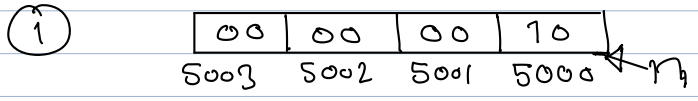
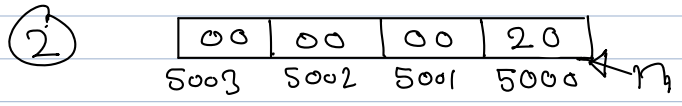
Advanced C Pointers
Day - I

Pointers : Absolute Fundamentals :

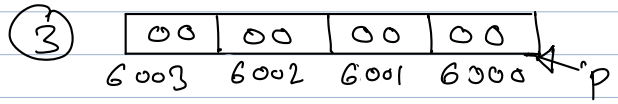
- To define data in C programming, a data definition statement is required. e.g. int n = 10;
- The data definition statement, not only allocates 4 bytes of memory to store value 10 but also names those 4 bytes so that we can read/write them.
- e.g. int n = 10;



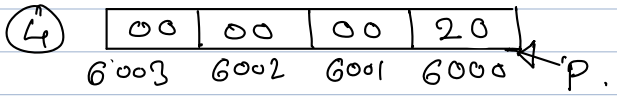
n = 20;



int p = 0;



p = n;



- If we can define data using the data definition statement and we can read/write on memory using variable name in the data definition statement then why do we need the pointer variable?
- Important Point :

Access to Any Variable Name in C is Restricted to the function in which it is defined. [except the global variable].

e.g.
```
void test(void)    /* CASE-I */
{
    int n = 10;
}
```

```
void func(void)
{
    n = 20;  /* Error : n defined in test()
                is not accessible func() */.
}
```

```
int n = 20;
void test()
{
    n = 30;
}
void func()
{
    n = 40;
}
```

/* CASE - II */.

/* As n is not defined in any function block, it is accessible every where in program. But it is not a good practice to keep all variables global */

So what if access of variable defined in function 1 is required by function 2?

e.g.

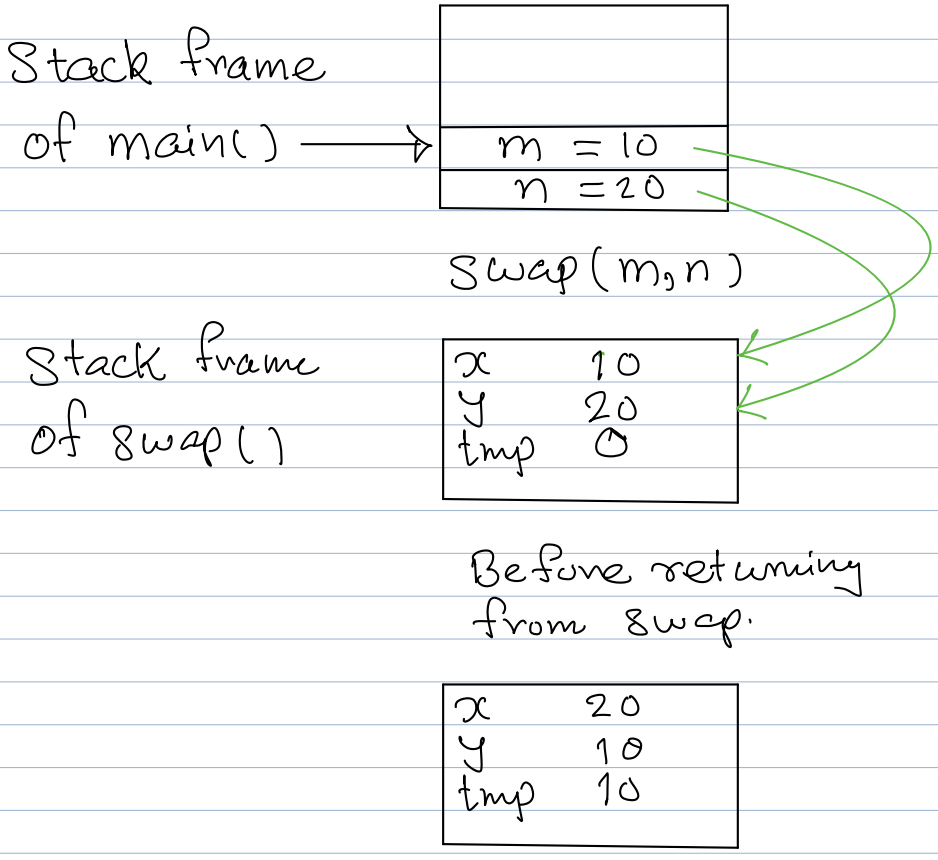
```
int main()
{
    int m=10, n=20.
    swap( , );
}
```

```
void swap(      )
{ /*-swap function does
    not have access to
    main() m & n
  - Also, w/o pointer main
    can send copies of m &
    n to swap which is
    not same as access
    to m & n
*/
}
```

Pass by value.

```
int main()
{
    int m=10, n=20;
    printf("m=%d, n=%d\n", m, n);
    swap(m,n);
    printf("m=%d, n=%d\n", m, n);
}
```

```
void swap(int x, int y)
{
    int tmp=0;
    tmp = x;
    x = y;
    y = tmp;
}
```



Note: 'm' and 'n' in are unaffected by the swap().
Copies of 'm' and 'n' were swapped & not 'm' & 'n' themselves.

- Basics of Indirect access.

Under C programming language, we can allocated the memory to store the address of another memory location. Such memory locations are known as 'pointers'.

- We've to use the following data definition statement to allocate pointer.

DataTypeName * PointerVariableName;

e.g. int * p;

float * fp;

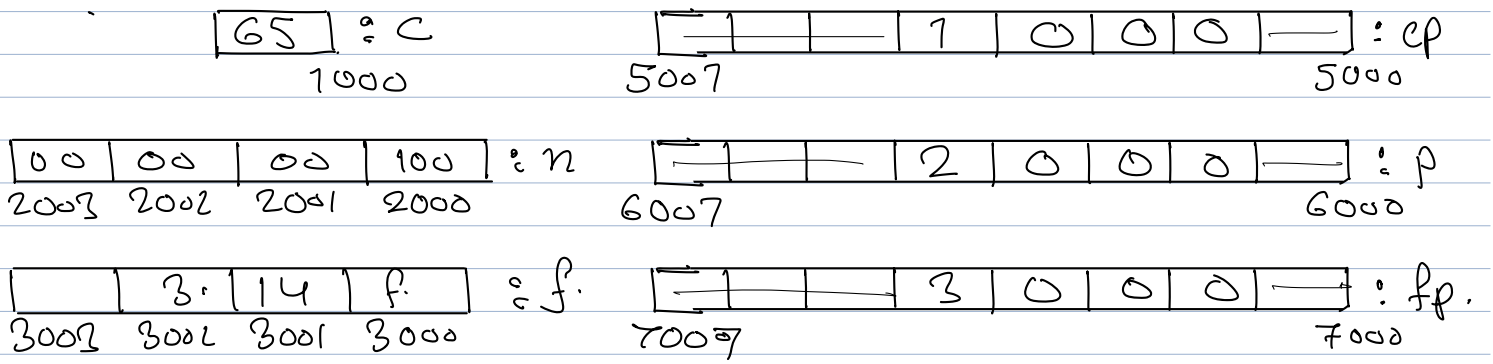
Char * cp;

- DataTypeName determines the type of data whose addresses can be stored in pointer variables.

e.g. char c = 'A'; char * cp = &c;

int n = 100; int * p = &n;

float f = 3.14f; float * fp = &f;



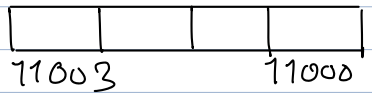
M[1000] ← 65 (ASCII('A'))	M[5000:5007] ← 1000
M[2000:2003] ← 100	M[6000:6007] ← 2000
M[3000:3003] ← 3.14	M[7000:7007] ← 3000.

Reading from pointers:

```
char c1 = *cp;
```

$M[9000] = M[\underbrace{M[5000:5007]}_{cp}] = M[1000]$ (*)

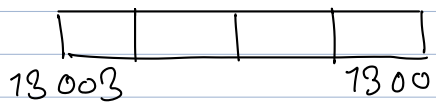
```
int n1 = *p;
```



$M[11000:11003] = M[\underbrace{M[6000:6007]}_{p}]$ (*)

$M[11000:11003] \leftarrow M[2000:2003]$.

```
float f1 = *fp
```

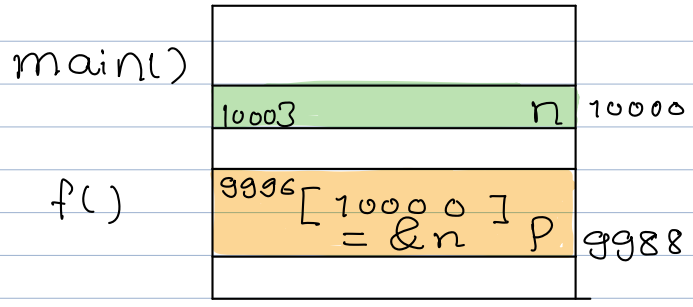


$M[13000:13003] \leftarrow M[\underbrace{M[7000:7007]}_{fp}]$ (*)

$M[13000:13003] \leftarrow M[3000:3003]$.

`int *p = &n;`

FORMAL PARAMETER — ACTUAL PARAMETER.




```
int main(void)
{
    int n = 100;
    f(&n);
    printf("n= %d\n", n);
}

void f(int *p)
{
    *p = 200;
}
```

1) $M[10000:10003] = n$

2) $M[9988:9988] = p$

3) $p = \&n \Rightarrow M[9988:9988] == 10000$

4) `*p = 200;`

$M[M[9988:9996]] \leftarrow 200$

$M[10000:10003] \leftarrow 200$.

Exercise - 1: Draw a diagram as shown above for following code.

```
int main()
{
    int m=10, n=20;
    printf("1: m=%d, n=%d\n", m,n);
    swap(&m, &n);
    printf("1: m=%d, n=%d\n", m,n);
    return (0);
}
```

```
void swap(int *x, int *y)
{
    int tmp;
    tmp = *x;
    *x = *y;
    *y = tmp;
}
```

```
/* x = Address of m
   y = Address of n
*/
/* Thus, we've solved
   the problem
*/
```