Module 4 :

Unit 1: Pointer
Unit 2: Pointer Assignment
Unit 3 : Pointer Arithmetic
Unit 4 : Advantages

Module 3 objective :

After completion of the module , the students will be able to

Objective 3.1 : Understanding – *explain* advantages of pointer

Objective 3.2 : Applying – *solve* problems efficiently using pointer

Objective 3.3 : Evaluating – *justify* the effectiveness of the code segments using pointers

By Value And By Address

Man Tom;

House h;
Color of Tom's Shirt = white ;
h = &Tom;
Color of the shirt of the person who lives in h (*h)= blue;

What is the color of Tom's shirt ?
What is the most important line in above example ?

By Value And By Address

Man T , M;

House h;
Color of T's Shirt = white ;
Color of the shirt of the person who lives in h (*h)= blue;

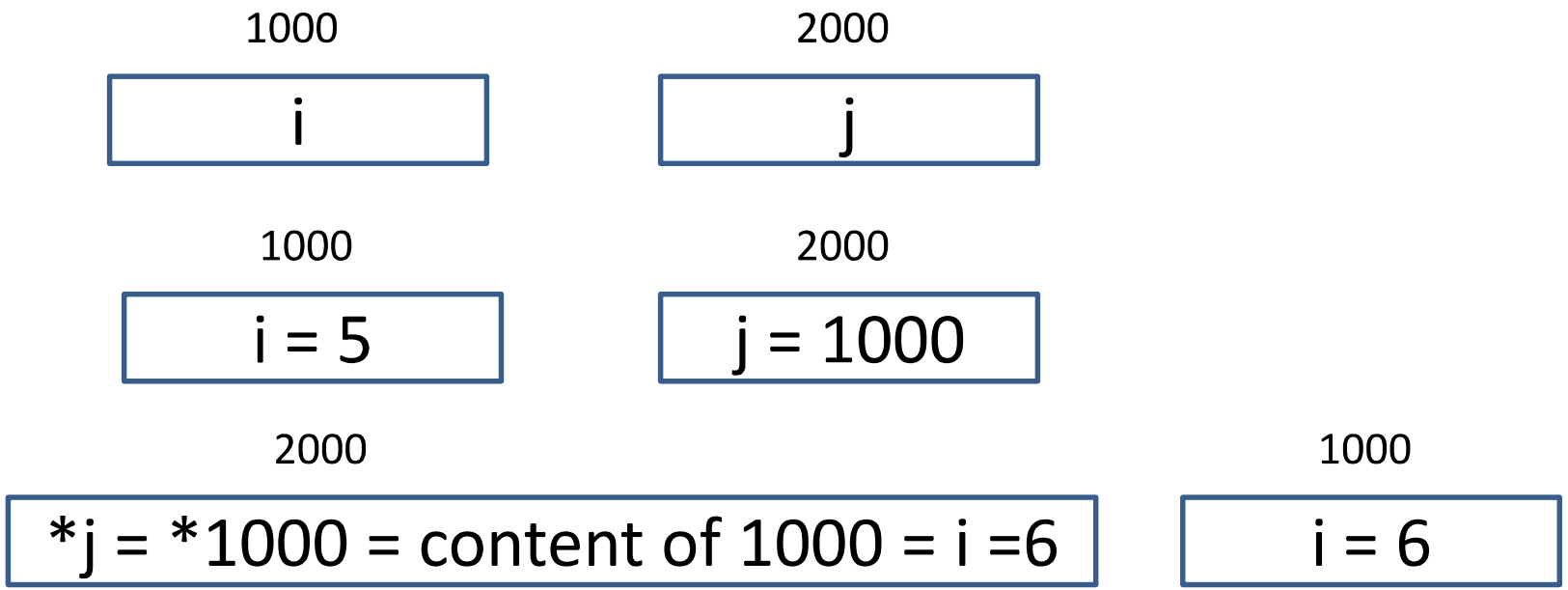
What is the color of T's shirt ?
What is the color of M's shirt ?

By Value And By Address

```
main(){ int i, *j;
   i = 5;
   *j = 6;
    printf ( “%d %d” , i, *j); }
```

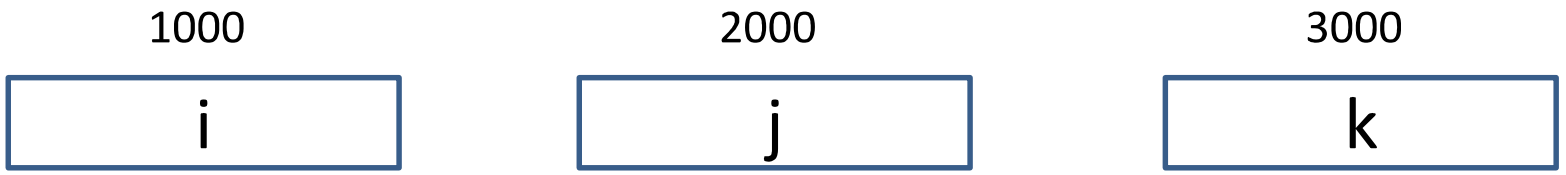
By Value And By Address

```
main(){ int i, *j;
   i = 5;
   j = &i;
   *j = 6;
    printf ( “%d %d” , i, *j); }
```



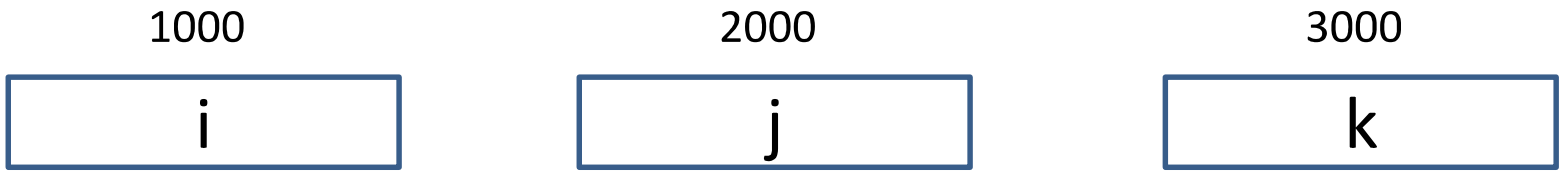
By Value And By Address

```
main(){ int i, *j , **k;
    i = 5;
    j = &i;
    *j = 6;
   k= &j;
  **k= 9;
    printf ( “%d %d %d %d %d %d %d %d %d” , i, *j, **k, &i, &j, &k, j, k,*k); }
```



By Value And By Address

```
main(){ int i, *j , **k;
    i = 5;
    j = &i;
    *j = 6;
   k= &j;
  **k= 9;
    printf ( “%d %d %d %d %d %d %d %d %d” , i, *j, **k, &i, &j, &k, j,k,*k); }
```



According to question

i = *j = **k ; j = *k ;

so answer is 9 9 9 1000 2000 3000 1000 2000 1000

By Value And By Address

```
main(){ int  *point , arr [10] , count;

        for (count =0; count <10; count++)

        arr[count] = count;

        point = arr ;

        for (count =0; count <10; count++)

        printf( “%d “, point[count] ); }
```

By Value And By Address

char arr[5]; // arr = 1000
We can tell

&arr[0] = arr + 0 (1000)	arr[0] = *(arr + 0)
&arr[0] = arr + 1 (1001)	arr[1] = *(arr + 1)
&arr[0] = arr + 2 (1002)	arr[2] = *(arr + 2)
&arr[0] = arr + 0 (1003)	arr[3] = *(arr + 3)
&arr[0] = arr + 0 (1004)	arr[4] = *(arr + 4)

int arr[5]; // arr = 1000
We can tell

&arr[0] = arr + 0 (1000)	arr[0] = *(arr + 0)
&arr[0] = arr + 1 (1002)	arr[1] = *(arr + 1)
&arr[0] = arr + 2 (1004)	arr[2] = *(arr + 2)
&arr[0] = arr + 0 (1006)	arr[3] = *(arr + 3)
&arr[0] = arr + 0 (1008)	arr[4] = *(arr + 4)

By Value And By Address

```
main(){ int  *point , arr [10] , count;

        for (count =0; count <10; count++)

        arr[count] = count;

        point = arr ;

        for (count =0; count <10; count++)

        printf( "%d ", count[point] );        }
/* point[count] = *( point + count )
   = * ( count + point ) = count [point]
but only variables are allowed no constant
 10[point] is not allowed */
```

Advantages of POINTER

1. Value of local variables can be changed through passing by address without declaring it to be global

```
int main(){ void adv_ptr1( int *);
            int i = 9;
            printf(" before calling function i = %d", i);
            adv_ptr1( &i );
            printf(" after calling function i = %d", i);
            return 0;}

void adv_ptr1( int *k){ (*k)++ ;
          /* *k++ will give wrong result */}
```

Advantages of POINTER

2. Dynamic Allocation of Memory

a) Creation of array in run time

```
main(){   int *arr,   size;
          scanf (“%d”, &size);
          arr = (int *) malloc ( size * sizeof(int));}
 /*   arr[size] is created */
```

Advantages of POINTER

2. Dynamic Allocation of Memory

b) Returning local array from functions

```
main() { int * fn() , * arr,  count;
         arr =  fn();
         for(count=0; count <10; count ++) printf("%d", arr[count]);}
```

/* Wrong Coding */

```
int * fn(){ int loc_arr[10], count;

for (count=0; count<10; count ++)
  loc_arr[count] = count;
   return loc_arr ; }
```

/* Correct Coding */

```
int * fn(){ int *loc_arr, count;

loc_arr = (int *) malloc (10 * sizeof(int));

for (count=0; count<10; count ++)
  loc_arr[count] = count;
   return loc_arr ; }
```