Review on Parameter Transmission

1/11/2023 ACTIVITY

GIVEN:

```
void exchange (int x, int y){
    int temp;

    temp = x;
     x   = y;
     y   = temp;
}
```

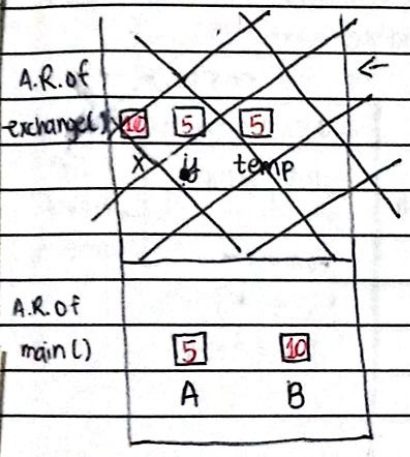
Function call:

```
int A=5;
int B=10;
exchange (A,B);
```

Show using execution stack why the function does not work. ASSUMPTION: Function call is in main()

Execution Stack: Each variable is illustrated using a box labeled with name, beginning address and value if there is any.

ANSWER:



PROBLEM!
no change in calling function

Problem in Scope & Lifetime because the variables in the calling function didn't change after the closing of the function exchange().

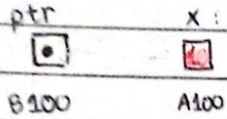
SOLUTION: Make the variables in main() accessible by the exchange function through pass by address so that change can be applied.

1/11/2023 ACTIVITY

GIVEN

```
int x = 10;
int * ptr;
```

Illustrate the declaration & initialization by drawing a box for each variables & label it with a name, beginning address and a value (if it exists)

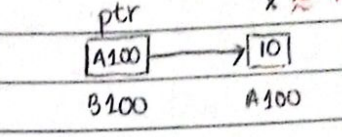


CONCEPT WISE QUESTION & ANSWER

1. what statement will let ptr point to x?

STATEMENT: ptr = &x;

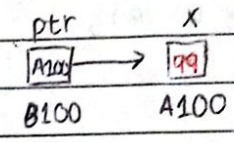
ILLUSTRATION:



2. What statement will change the value to 99 using ptr?

STATEMENT: *ptr = 99;

ILLUSTRATION:



exchange using pointers (pass by address)

1. Write the function header

void exchange (int * x, int y)

2. Write the appropriate function call. Declare the variables before the call and initialize them if necessary

```
int A = 5;
int B = 10;
exchange (&A, &B); -> function call
```

3. Assume that the function call is in main, draw the execution Stack. Illustrate.

4. Write the code of the function. Refer to execution stack to correctly access variable

WRONG

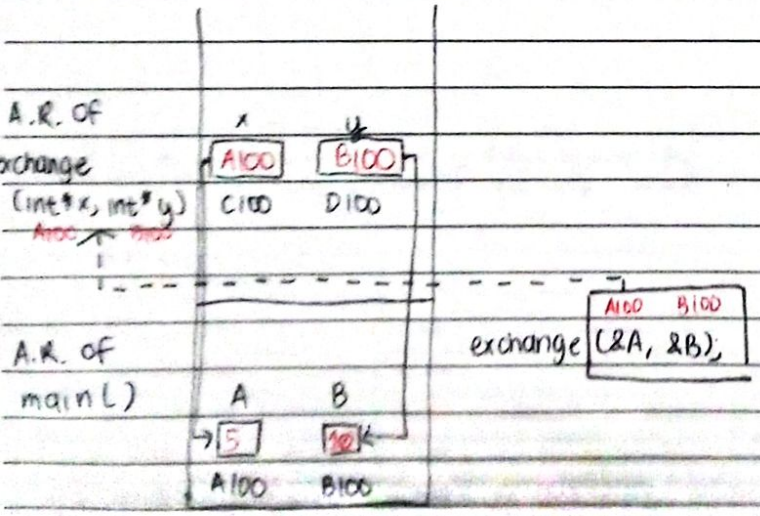
```
void exchange (int *x, int *y){
int * temp;          -> TYPE MISMATCH (should get address)
temp = *x;
*x = *y;
*y = *temp;
```

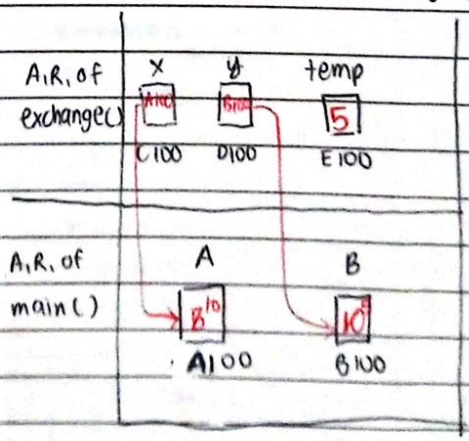
SOLN: Not Pointer (temp)

CORRECT

```
void exchange (int*x, int*y)
int temp;            -> Applied solution
temp = *x;
*x = *y;
*y = temp;
```

5. Simulate the code (for the correct ☜)





AGGREGATE

Structures Exchange

11/11/2023 ACTIVITY

GIVEN

Data type definition

```
typedef struct node {
    int x;  x-axis
    int y;  y-axis
} Point;
```

Function Specification : The function will exchange the values of 2 given points of the cartesian coordinate

1. Write the function header

void exchange (Point * x, Point * y)

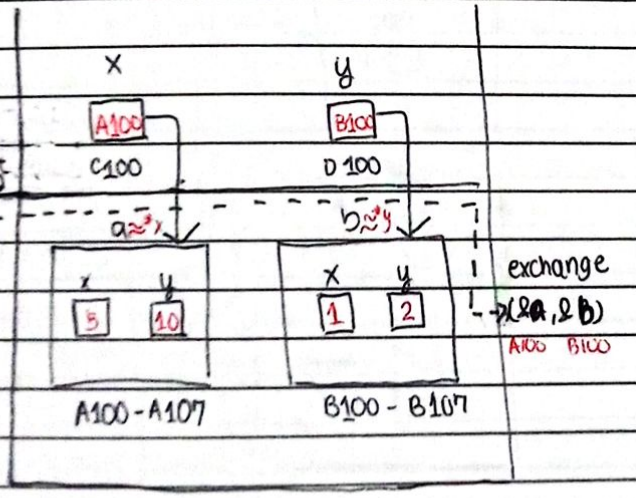
2. Write the appropriate function call. Declare the variables before the call and initialize them if necessary

```
Point a = { 5, 10 };   //you can initialize this
Point b = { 1, 2 };    instant ra basta in-order
```

x-axis y-axis

exchange (&a, &b); → Function call

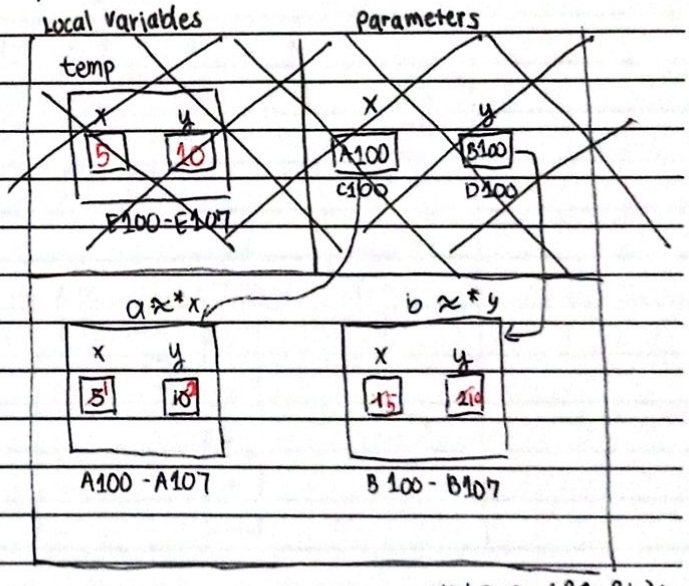
3. Assume that the function call is in main, illustrate by drawing the execution stack.



4. Write the code of the function. Refer to execution stack to correctly access variable.

```
void exchange (Point * x, Point * y) {
    Point temp;
    temp = * x;
    *x = *y;
    *y = temp;
}
```

5. Re-simulate to check if sakto ^ω^



JANUARY 19 2022 EXERCISE

Data Structure Definition

```
typedef struct{
    char FN [24];
    char LN [16];
    char MI;
} Nametype  //41 bytes
```

Problem specification : The function will exchange the values of 2 given variables of type Nametype

Do the ff:

1. Write the function header

```
void exchange ( Nametype * x , Nametype* y)
```

2. Write an appropriate function call. Declare the variables before the call and initialize if necessary

```
Nametype sab = {"Sabrinan Yonell","Yap",'C'};
Nametype miss = {"Christine","Pena",'C'};
exchange (&sab, &miss);   → FUNCTION CALL
```

Scalar within Array

JANUARY 20, 2023

Problem Specification : The function will accept as parameter an array of integers and an actual number of integers in the array, and an integer x. The function will return TRUE if x is in the given array, otherwise return FALSE.

CONSTRAINT: Only 1 Return statement, No break and Exit.

① Function Header:

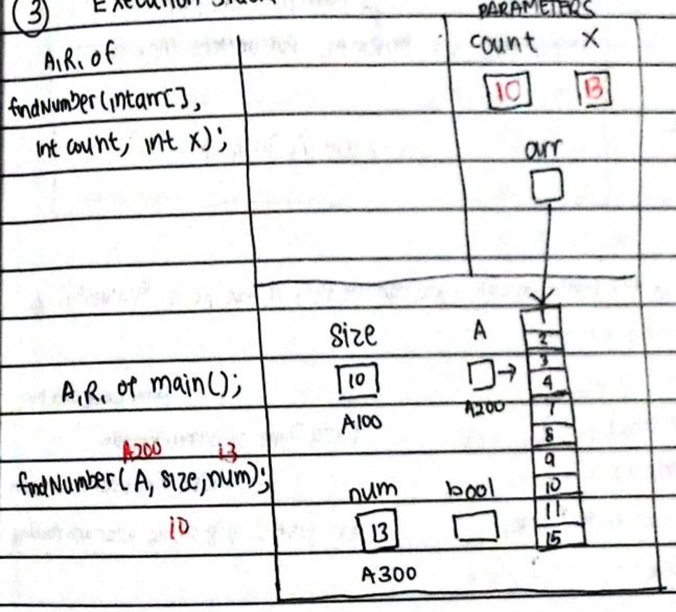
```
boolean findNumber (int arr[], int count, int x);
```

better in coding convention, because if you use *, compiler will be accepting / expecting an address of integer

② Function Call:

```
int size = 10;
int A[size] = {1, 2, 3, 4, 7, 8, 9, 10, 11, 15};
int num = 13;
boolean bool = findNumber (A, size, num); //Function Call
```

③ Execution Stack



④ Code of the function

```
boolean findNumber (int arr[], int count, int x) {
    int ctr;
                        these 2 conditions should not interchange
    for (ctr=0; ctr<count && arr[ctr]!=x ; ctr++) {}
⊗   return (arr[ctr]==x)? TRUE: FALSE;
```

↳ THIS IS ACTUALLY WRONG! There's a tendency that you'll go beyond the array size.

```
    return (ctr < count)? TRUE : FALSE;
```

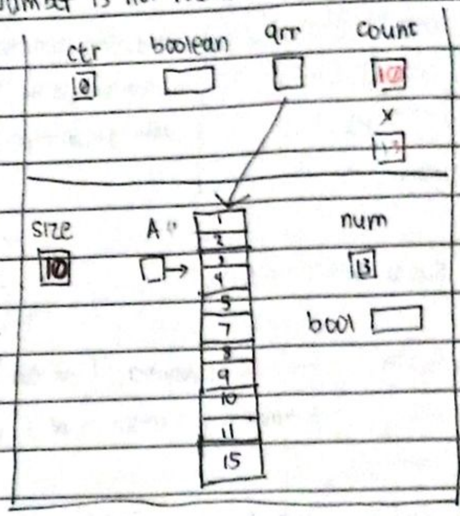
↳ THIS MAY NOT MAKE SENSE AT FIRST because you must find a number, but this is the correct answer.

NOTE: SAB STOP overthinking ✕✕

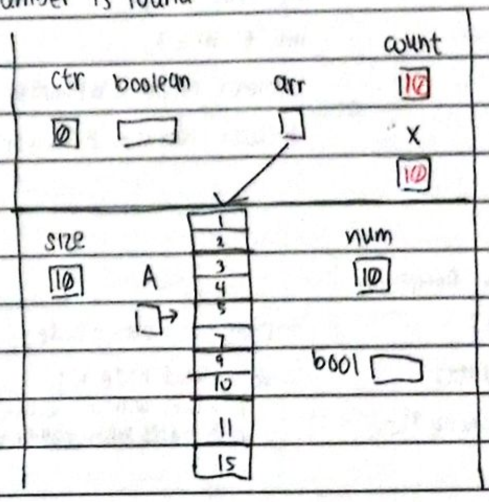
```
}
```

⑤ Visualizing Test Cases

ⓘ If Number is not found



ⓘⓘ If Number is found



BONUS! Do this but for Linked List implementation

```
boolean findElem (List L, char elem) {
    List trav;
    for (trav=L; trav != NULL && trav->data != elem; trav=trav->link) {}
    return (trav->link != NULL) ? TRUE : FALSE;
}
```

^^ see how this is almost the same for arrays ^^

Limelight

JANUARY 24, 2023

Summary / Review

EXERCISE 1: void exchange (int x, int y)

» pass by value (copy)

» Execution stack CONCLUSION: Does not work

EXERCISE 2: void exchange (int *x, int y)

» Pass by address

» Execution stack illustrating pointers to a scalar variable

EXERCISE 3: Aggregate Data Type (Structure)

```
typedef struct {
    char FN[24];
    char LN[16];
    char MI;
} Nametype
```

Initializing structure & array should be together ⇒ with declaration.

CORRECT: `Nametype N = {"Bruce", "Alturas", 'A'};`

WRONG:
```
Nametype N;
N = {"Bruce", "Alturas", 'A'};
```

EXERCISE 4: Scalar within an Array

A) Function Header: `boolean IsMember (int arr[], int size, int x)` (FORMAL PARAMETERS / ARGUMENTS)

// function will return TRUE if x is in arr, otherwise return FALSE

B) Function Call:
```
int A[5] = {1,2,3};
int size = 5;
int elem = 3;
boolean retval = isMember (A, size, elem);
```
(ACTUAL PARAMETERS)

or

```
boolean retval = isMember (A, 5, 3);
```

literal constants in the function call, you can use this if you pass "values" only.

FUNCTION HEADER

ACTUAL vs FORMAL Parameters (Arguments)

FUNCTION CALL

These are PL Terms, btw ^^

Data Type Definition

```
struct node{
    char data;
    struct node *link;
};
```

Data Type: struct node;

ex. struct node N;

// 9 Bytes without padding

// 16 Bytes with padding

vs

```
typedef struct node{
    char data
    struct node * link;
} * List
```

Data type: struct node (→ structure data type)

List → pointer to a struct

ex. List L // 8 Bytes w/ or w/o padding

JANUARY 26, 2023

Review on Aggregates

"typedef"
- ↳ definition
- ↳ data type

so if you combined both, "Data Type definition"

STRUCTURE

studtype stud

name	FName	// 24 characters
	LName	// 16 characters
	MI	
ID	//integer	
course	// 6-characters	
yearLevel	// integer	

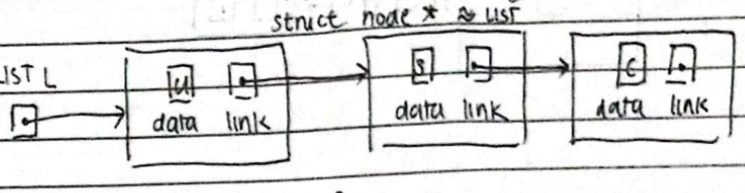
Data Type definition

```
typedef struct {
    char FName[24];
    char LName[16];
    char MI;
} Nametype;
```

```
typedef struct{
    Nametype name;
    int ID;
    char course[8];
    int yearLevel;
} studtype
```

LINKED LIST

struct node ** ≈ LIST *
struct node * ≈ celltype
struct node * ≈ LIST




```
typedef struct node {
    char data;
    struct node *link;
} nodetype;
```

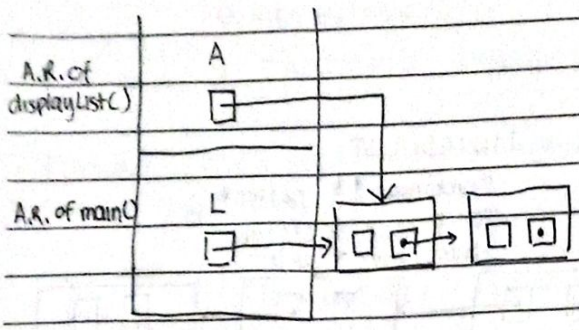
↳ struct node jud ni sya.
you cannot use nodetype* because compiler does not know who is nodetype yet.

COMPARING

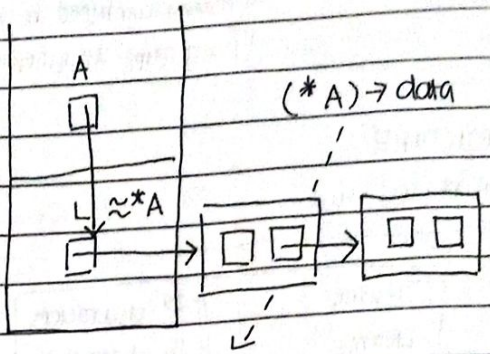
	Self-referenced Structures	"Typedef" Structures
code snippet	struct node { char data; struct node * link; } x;	typedef struct node { char data; struct node * link; } y;
data type	struct node	struct node
variable	x	y
To declare more variables	struct node A, *B;	none
To define additional data type	typedef struct node celltype, * LIST;	⇒ same ra pero use y if you want instead of struct node

CONTINUE

PASS BY VALUE (COPY)

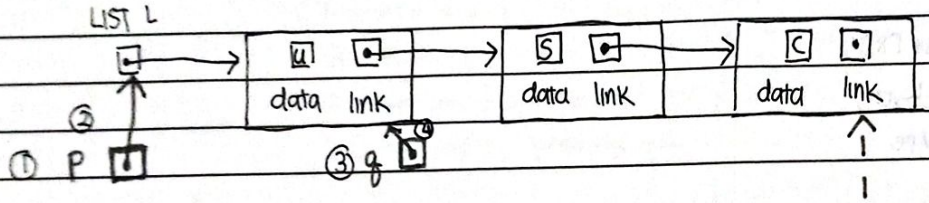


PASS BY ADDRESS (REFERENCE)



need () for * A because → has a higher precedence so if theres none, a→data will be accessed first, which is not the accessing we aim

TRAVERSALS (use p as PPN (Pointer to Pointer to Node), q as PN (Pointer to Node))




```
LIST L ;   //ASSUME: Populate with 3 elements
①  LIST * p ;   // declare p
②  p= &L ;   // let p point to L
③  LIST q ;   // declare q
④  q= L ;   // let q initialize to the first element of L
POINTER TO POINTER TO NODE TRAVERSAL  //Use this in delete & insert
⑤  for (p=&L ; *p!=NULL ; p= &(*p)→link) {} - - - -
POINTER TO NODE TRAVERSAL  //Use this as display
⑥  for (q= L ; q!= NULL ; q =q→link) {}
```

How many conditions needed in the loop?

Display: 1

If you need to find then stop : 2

Condition when traversing the linked list array

1) 1-Loop Condition → traversal will reach the end ex displayElem()
2) 2-Loop Condition → (&&) travel will not always reach the end
   ex searching for a certain element