

REFERENCES, POINTERS PASSING PARAMETERS TO FUNCTIONS

Problem Solving with Computers-I

C++

```
#include <iostream>
using namespace std;

int main(){
    cout<<"Hola Facebook\n";
    return 0;
}
```



How far along are you with lab04

- A. Almost done
- B. I am on track to finish
- C. I am stuck and don't know how to proceed
- D. Haven't started

Pass by value

```
void swapValue(int x, int y){  
    int tmp = x;  
    x = y;  
    y = tmp;  
}  
  
int main() {  
    int a=30, b=40;  
    cout<<a<<" "<<b<<endl;  
    swapValue( a, b);  
    cout<<a<<" "<<b<<endl;  
}
```

What is printed by
this code?

A.

30 40

30 40

B.

30 40

40 30

C. Something else

References in C++

```
int main() {  
    int d = 5;  
    int &e = d;  
}
```

A reference in C++ is an alias for another variable

References in C++

```
int main() {  
    int d = 5;  
    int & e = d;  
    int f = 10;  
    e = f;  
}
```

How does the diagram change with this code?

A.  d: [5]
e: [10]

B.  d: [5]
e: [10]
f: [10]

C.  d: [10]
e: [10]
f: []

D. Other or error

Pointers and references: Draw the diagram for this code

```
int a = 5;  
int &b = a;  
int *pt1 = &a;
```

What are three ways
to change the value of
'a' to 42?

Passing parameters by reference

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

```
int main() {  
    int a=30, b=40;  
    swapValue(a, b);  
    cout<<a<<" "<<b<<endl;  
}
```

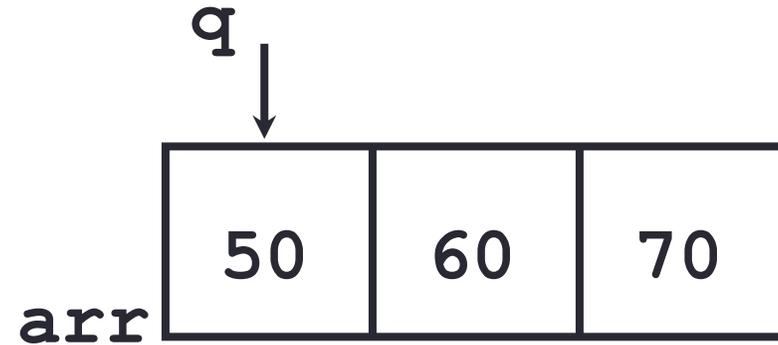
Passing parameters by address

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

```
int main() {  
    int a=30, b=40;  
    swapValue( a, b);  
    cout<<a<<" "<<b<<endl;  
}
```

```
void IncrementPtr(int *p){  
    p++;  
}
```

```
int arr[3] = {50, 60, 70};  
int *q = arr;  
IncrementPtr(q);
```



Which of the following is true after **IncrementPtr (q)** is called in the above code:

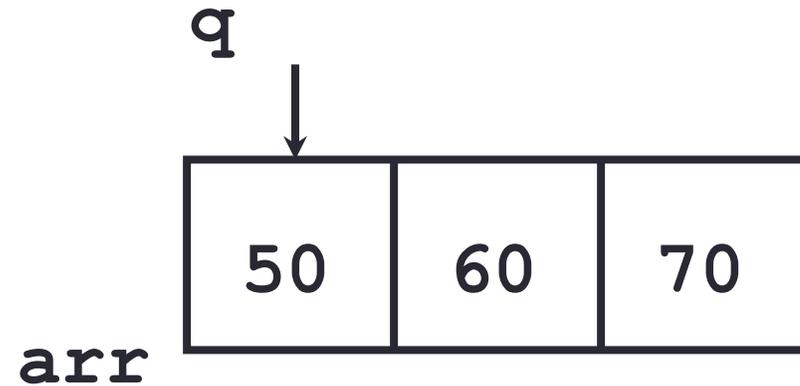
- A. 'q' points to the next element in the array with value 60
- B. 'q' points to the first element in the array with value 50

How should we implement `IncrementPtr()`, so that 'q' points to 60 when the following code executes?

```
void IncrementPtr(int **p){  
    p++;  
}
```

```
int arr[3] = {50, 60, 70};  
int *q = arr;  
IncrementPtr(&q);
```

- A. `p = p + 1;`
- B. `&p = &p + 1;`
- C. `*p = *p + 1;`
- D. `p = &p+1;`



Two important facts about Pointers

1) A pointer can only point to one type –(basic or derived) such as `int`, `char`, a `struct`, another pointer, etc

2) After declaring a pointer: `int *ptr;`
`ptr` doesn't actually point to anything yet.

We can either:

- make it point to something that already exists, OR
- allocate room in memory for something new that it will point to
- Null check before dereferencing

Pointer Arithmetic

- What if we have an array of large structs (objects)?
 - C++ takes care of it: In reality, `ptr+1` doesn't add 1 to the memory address, but rather adds the size of the array element.
 - C++ knows the size of the thing a pointer points to – every addition or subtraction moves that many bytes: 1 byte for a char, 4 bytes for an int, etc.

Pointer pitfalls

- Dereferencing a pointer that does not point to anything results in undefined behavior.
- On most occasions your program will crash
- Segmentation faults: Program crashes because code tried to access memory location that either doesn't exist or you don't have access to

Next time

- Structs
- Arrays of structs