

POINTERS AND REFERENCES

DYNAMIC MEMORY

Problem Solving with Computers-I

C++

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

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



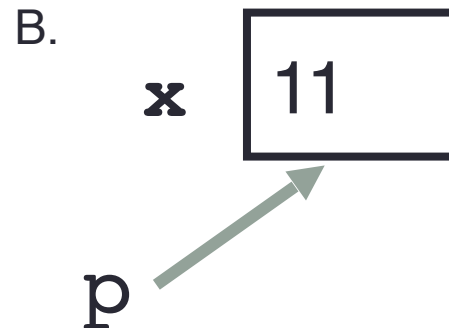
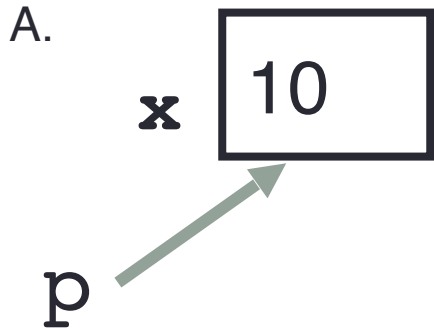
Learning Goals

- Understand pointer mechanics and how they are used to pass parameters to functions
- Creating data on the heap with new and delete
- Difference between data on the heap vs. data on the stack
- Functions returning pointers

Tracing code involving pointers

```
int* p;  
int x = 10;  
p = &x;  
*p = *p + 1;
```

Q: Which of the following pointer diagrams best represents the outcome of the above code?

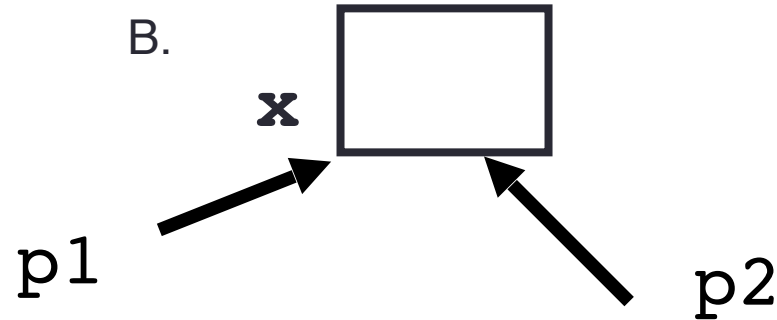
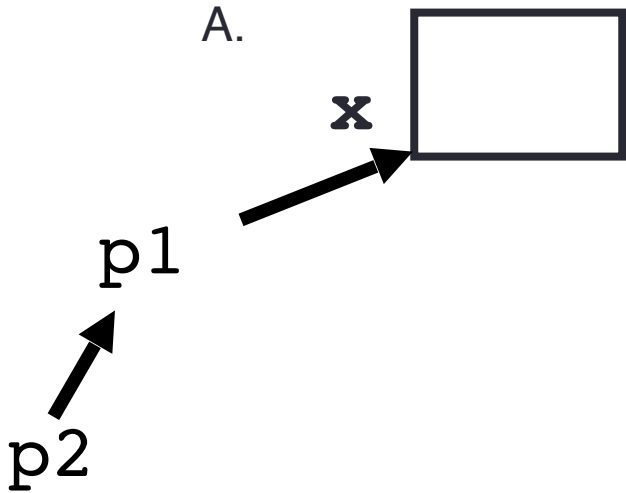


C. Neither, the code is incorrect

Pointer assignment

```
int* p1, *p2, x;  
p1 = &x;  
p2 = p1;
```

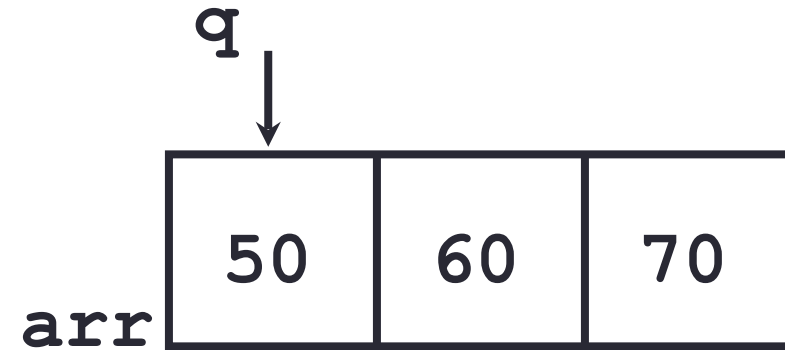
Q: Which of the following pointer diagrams best represents the outcome of the above code?



C. Neither, the code is incorrect

```
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:

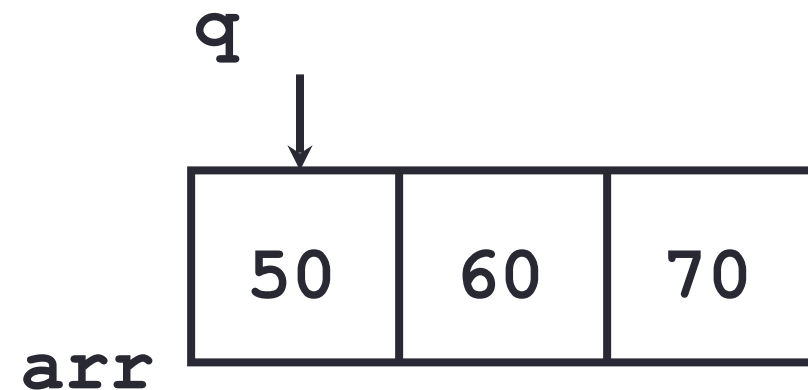
- A. 'q' points to the next element in the array with value 60
- B. 'q' points to the first element in the array with value 50

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;`



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

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:

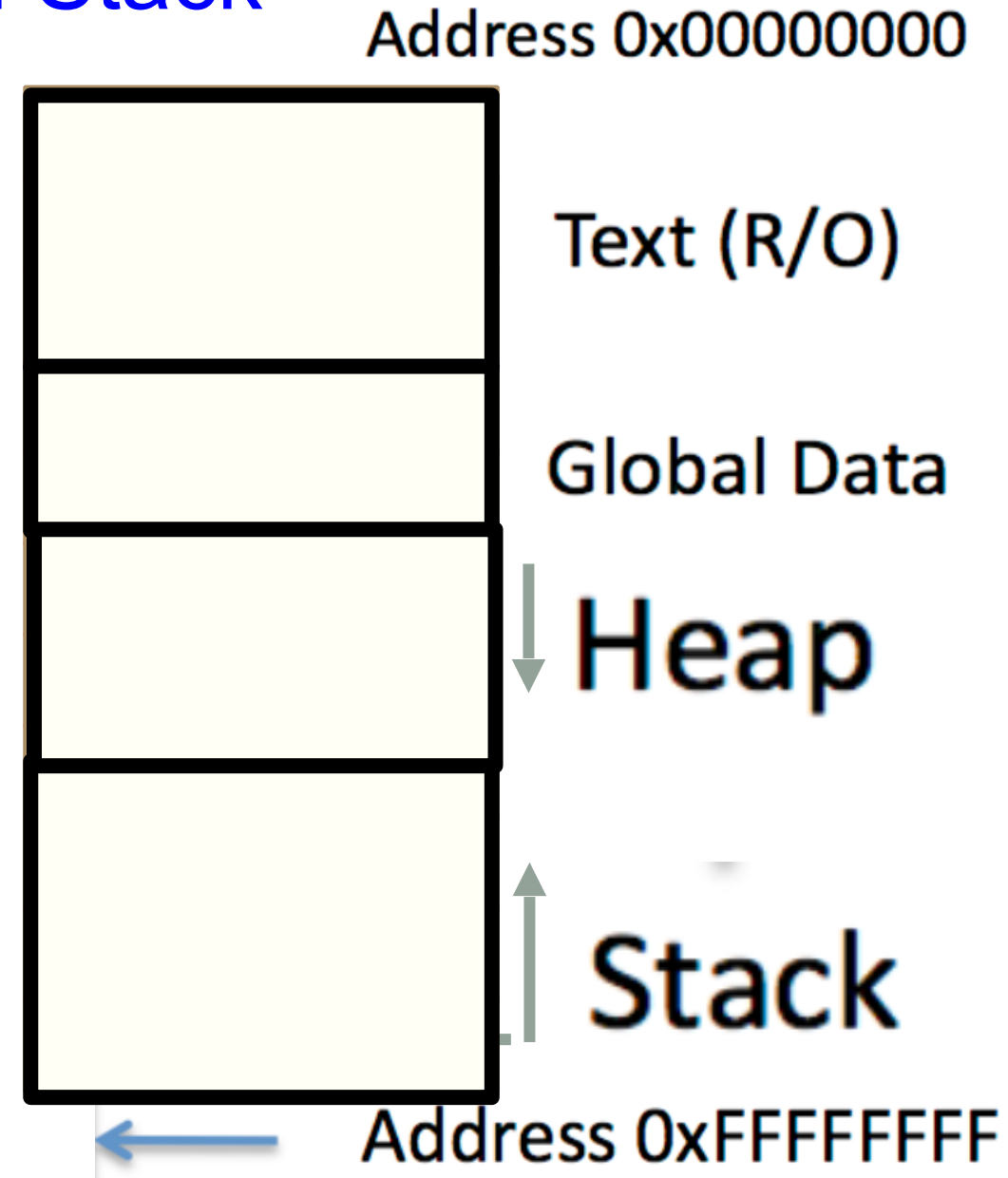
- make it point to something that already exists, OR
- allocate room in memory for something new that it will point to

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.

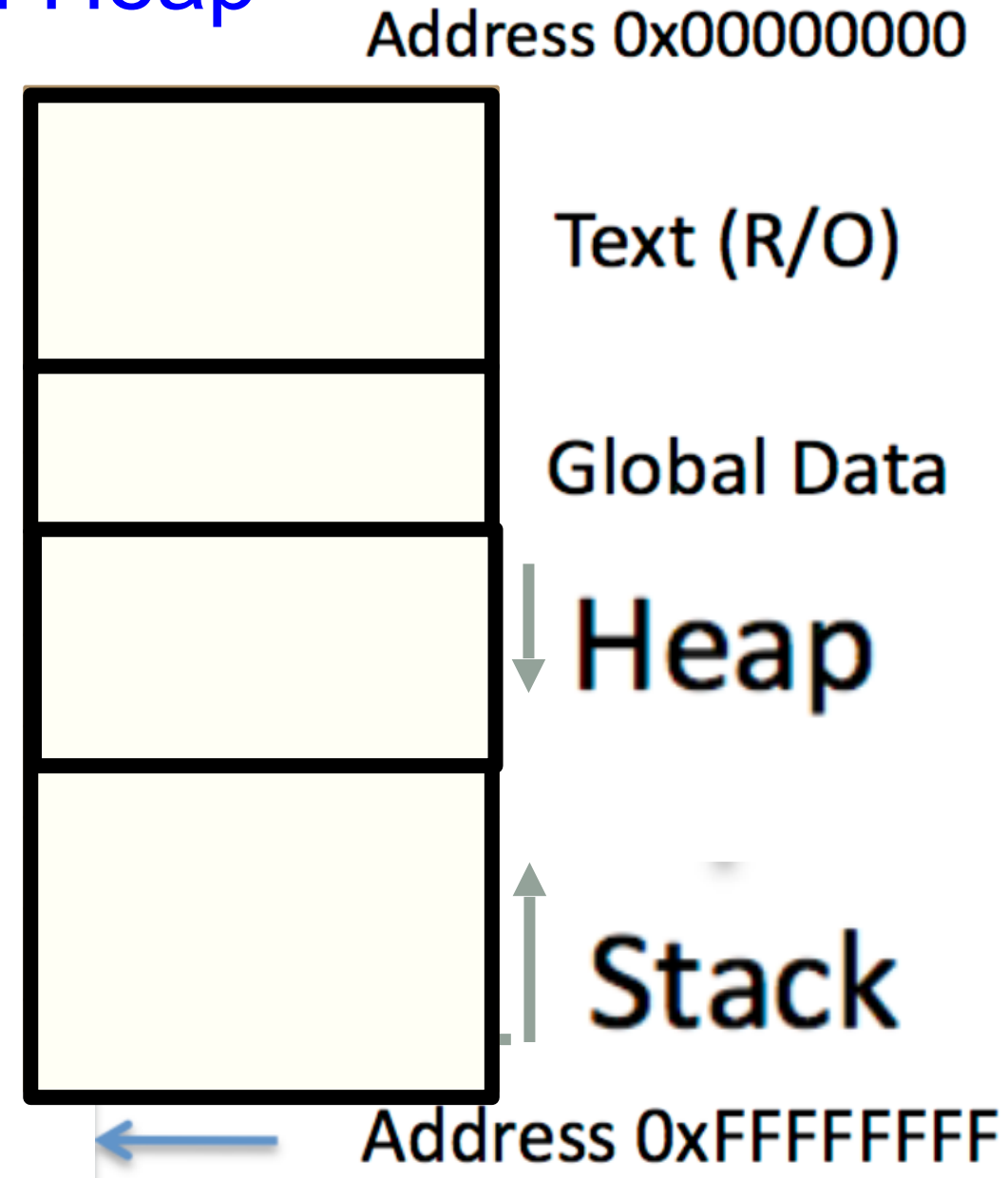
C++ Memory Model: Stack

- Stack: Segment of memory managed automatically using a Last in First Out (LIFO) principle
- Think of it like a stack of books!



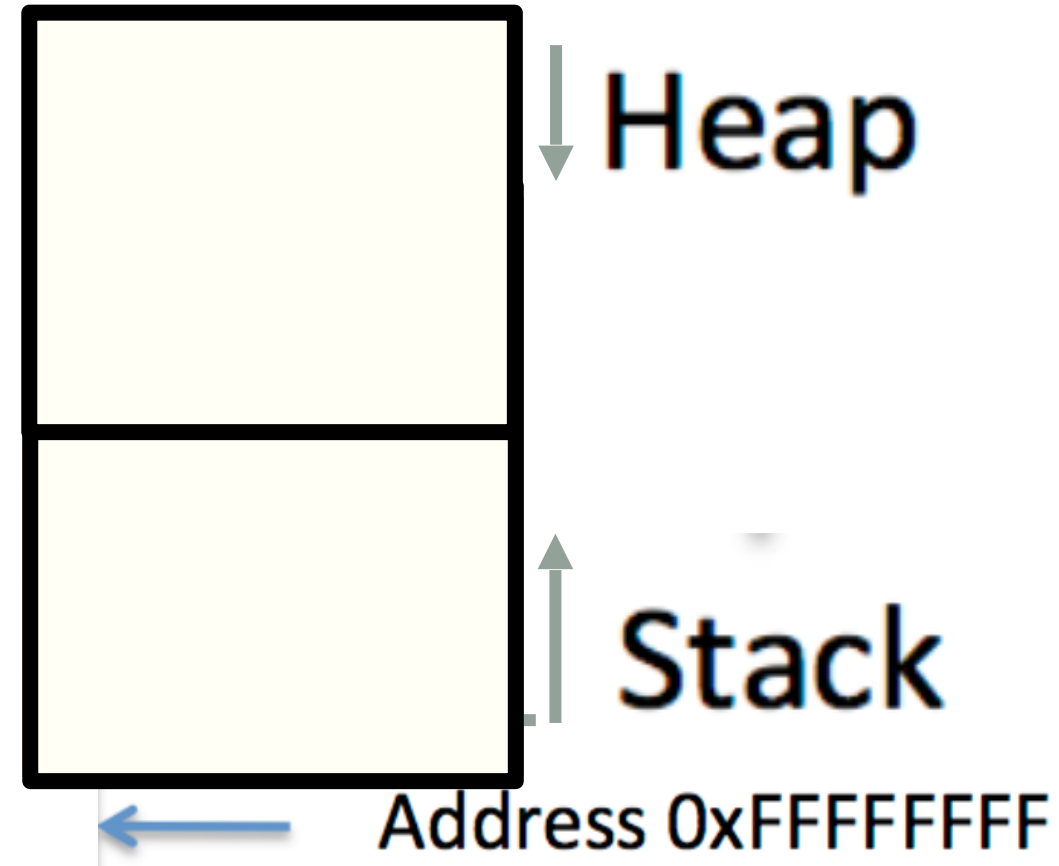
C++ Memory Model: Heap

- Heap: Segment of memory managed by the programmer
- Data created on the heap stays there
 - FOREVER or
 - until the programmer explicitly deletes it



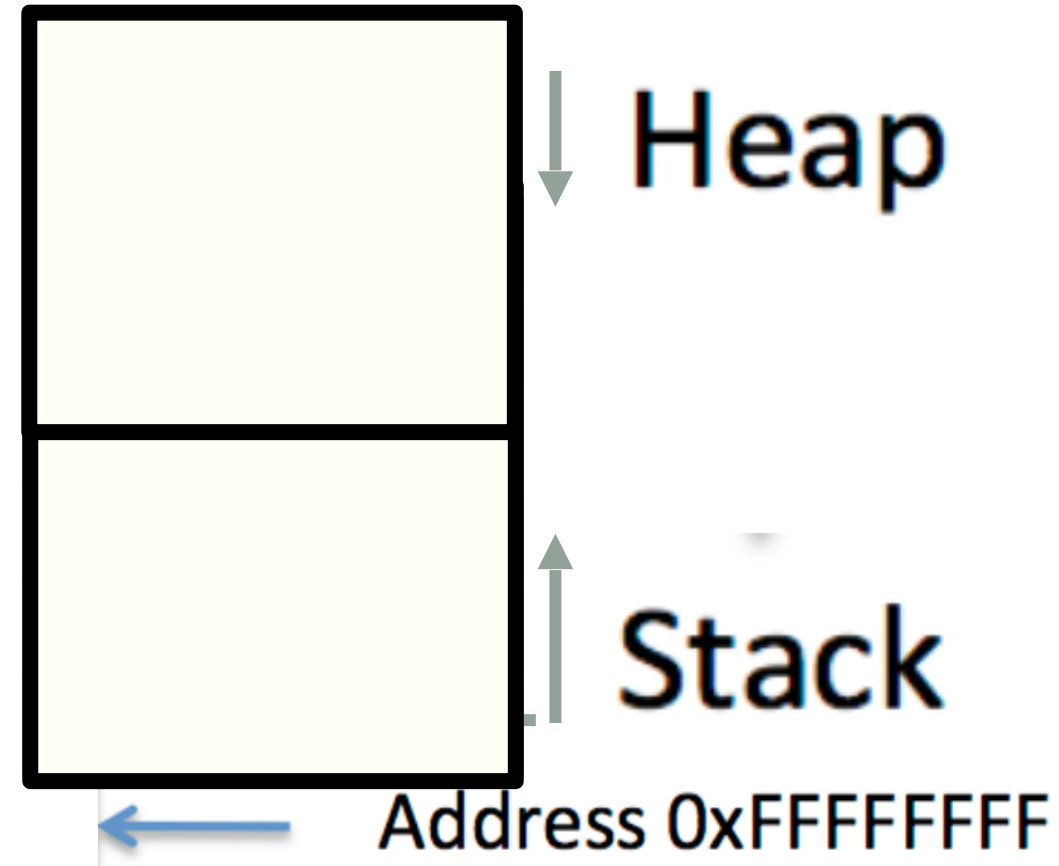
Creating data on the Heap: new

To **allocate** memory on the heap use the **new** operator



Deleting data on the Heap: delete

To **free** memory on the heap use the **delete** operator



Dynamic memory management = Managing data on the heap

```
int* p= new int; //creates a new integer on the  
heap
```

```
SuperHero* n = new SuperHero;  
                //creates a new Student on the  
heap
```

```
delete p; //Frees the integer
```

```
delete n; //Frees the Student
```

The case of the disappearing data!

```
int getInt() {
    int x=5;
    return x;
}
int* getAddressOfInt() {
    int x=10;
    return &x;
}
int main() {
    int y=0, *p=nullptr, z=0;
    y = getInt();
    p = getAddressOfInt();
    z = *p;
    cout<<y<<" , "<<z<<" , "<<*p<<endl;
}
```

What is the output?

A. 5, 0, 10

B. 5, 10, 10

C. Something else

Heap vs. stack

```
1 #include <iostream>
2 using namespace std;
3
4 int* createAnIntArray(int len){
5
6     int arr[len];
7     return arr;
8
9 }
```

Does the above function correctly return an array of integers?

A. Yes

B. No

Next time

- Rule of three and Linked Lists