POINTERS

Problem Solving with Computers-I

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

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

GitHub

Why learn pointers?

Pass by value: What is printed by this code?

```
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;
}
```

[✓ A.]

```
30 40
30 40
```

B.

```
30 40
40 30
```

C. Something else

Pointers

- Pointer: A variable that contains the address of another variable
- Declaration: *type* * pointer_name;

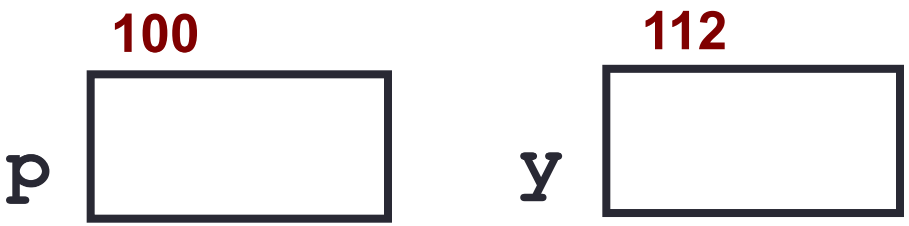
```
int* p;
```

// Just like all uninitialized variables this will have a junk value

```
int* p = 0; //Declare and initialize
```

How to make a pointer point to something

```
int *p;
int y =3;
```

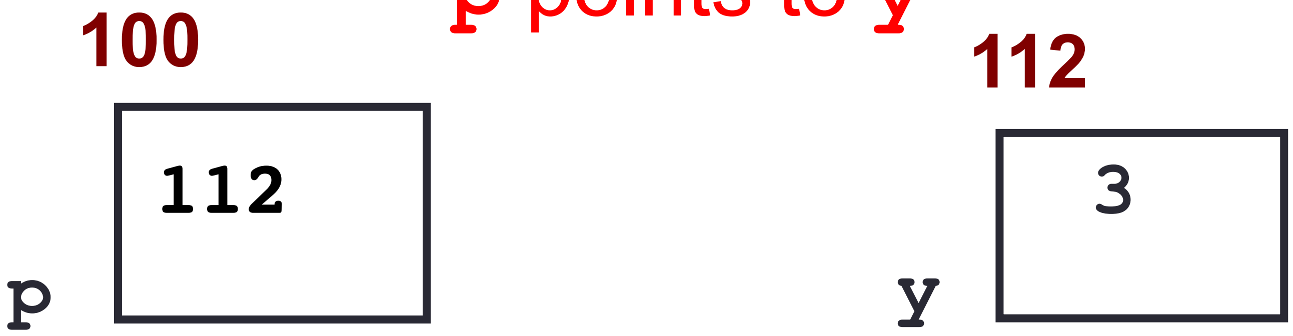


To access the location of a variable, use the address operator ‘&’

Pointer Diagrams:
Diagrams that show the relationship between pointers and pointees

p points to **y**

Pointer: p



100

112

p

112

3

y

Pointee: y

You can change the value of a variable using a pointer !

```
int *p, y;
y = 3;
p = &y;

*p = 5;
```

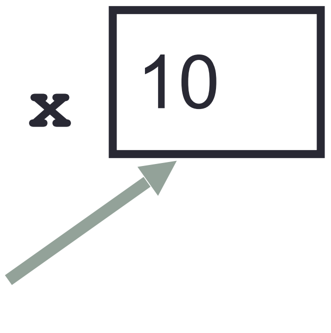
Use dereference * operator to left of pointer name

Tracing code involving pointers

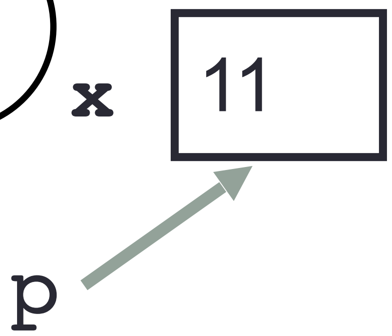
```
int *p;
int x=10;
p = &x;
*p = *p + 1;
```

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

A.

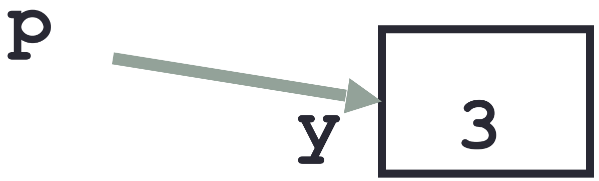


[✓ B.]



C. Neither, the code is incorrect

Two ways of changing the value of a variable



Change the value of y directly:

Change the value of y indirectly (via pointer p):

Pointer assignment and pointer arithmetic: Trace the code

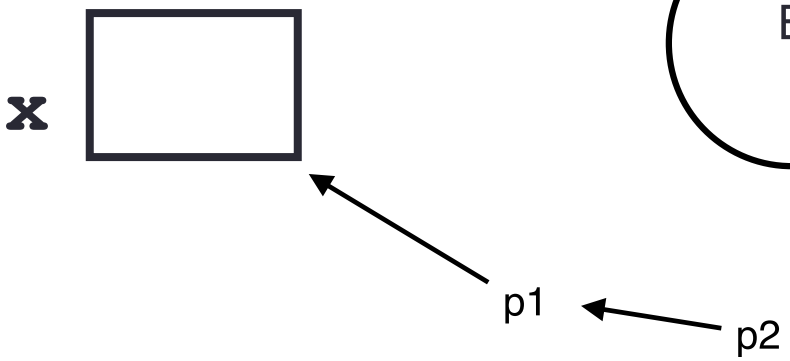
```
int x=10, y=20;

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

p2 = p1;

int **p3;

p3 = &p2;
```

Pointer assignment

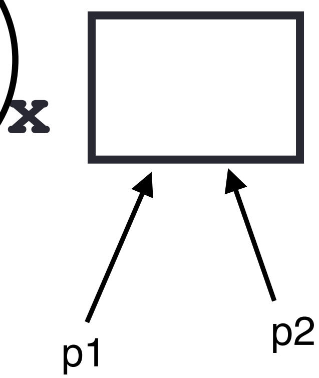
```
int *p1, *p2, x;
p1 = &x;
p2 = p1;
```

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

A.



[✓ B.]



C. Neither, the code is incorrect

Swap values revisited: Pass by address

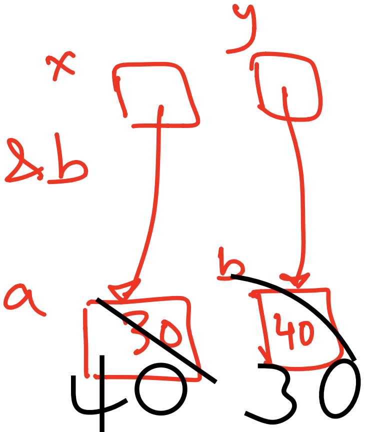
```
 void swapValue(int  x, int  y){
      int tmp = x;
      x = y;
      y = tmp;
 }
int main() {

     int a=30, b=40;

     swapValue( a,  b);

     cout<<a<<"  "<<b<<endl;

}
```

Swap values revisited: Pass by address

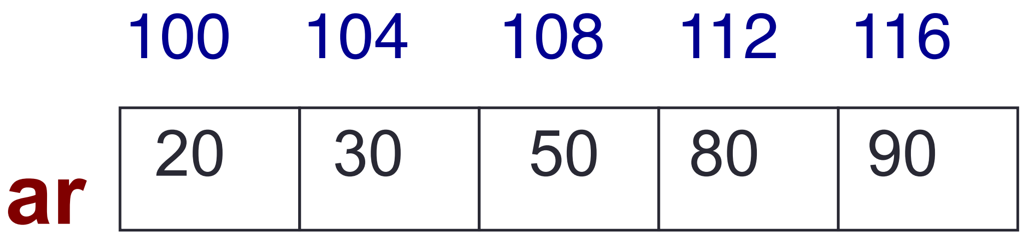
```
void swapValue(int* x, int*  y){
     int tmp =*x;
    *x =*y;
    *y = tmp;
}

int main() {

    int a=30, b=40;

    swapValue(&a,&b);

    cout<<a<<" "<<b<<endl;

}
```

→ swap the values of variables that x & y are pointing to. (In this case a,b)

// Pass the address of a & b



Arrays and pointers



	100	104	108	112	116
ar	20	30	50	80	90

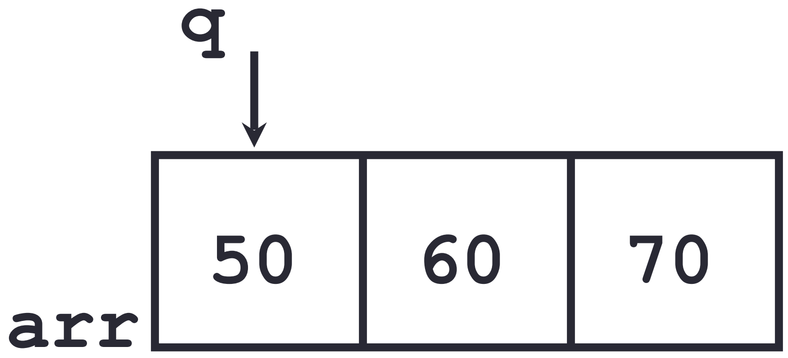
- `ar` is like a pointer to the first element
- `ar[0]` is the same as `*ar`
- `ar[2]` is the same as `*(ar+2)`

- Use pointers to pass arrays in functions
- Use *pointer arithmetic* to access arrays more conveniently

Pointer Arithmetic

```
int arr[]={50, 60, 70};
int *p;
p = arr;
p = p + 1;
*p = *p + 1;
```

```
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?

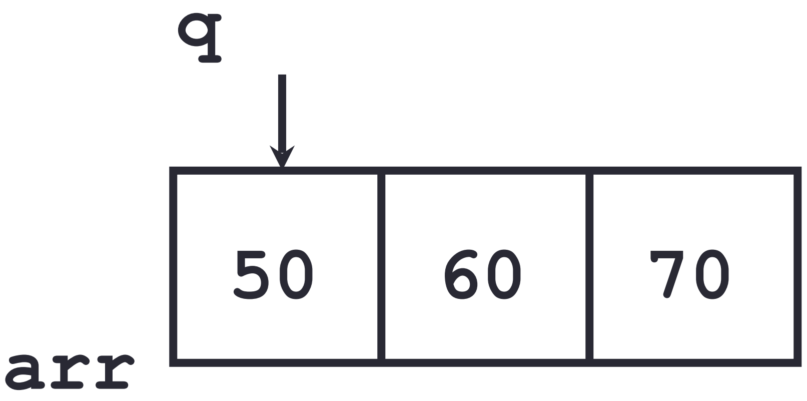
```
void IncrementPtr(int **p){
     p++;
}

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

Why learn pointers?…to get CS jokes

MAN, I SUCK AT THIS GAME. CAN YOU GIVE ME A FEW POINTERS?
0x3A28213A 0x6339392C, 0x7363682E.
I HATE YOU.