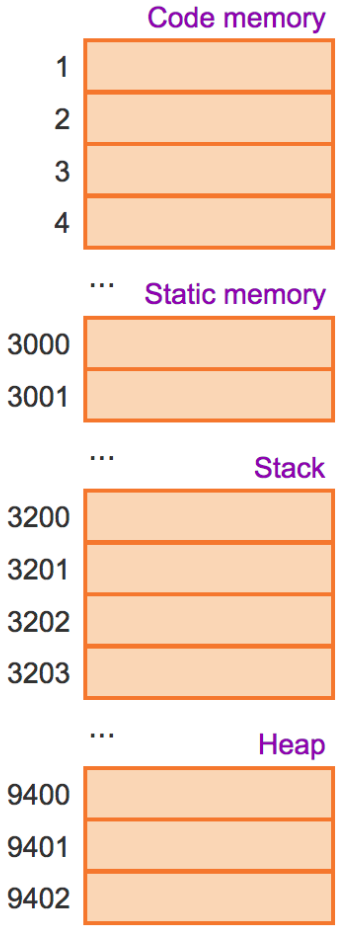
C++ PROGRAM MEMORY MODEL, POINTERS AND REFERENCES

Problem Solving with Computers-I

C++ Memory Model a.k.a Program's Memory Regions

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

// Program is stored in code memory

int myGlobal = 33;     // In static memory

void MyFct() {
   int myLocal;        // On stack
   myLocal = 999;
   cout << " " << myLocal;
}

int main() {
   int myInt;             // On stack
   int* myPtr = nullptr; // On stack
   myInt = 555;

   myPtr = new int;       // In heap
   *myPtr = 222;
   cout << *myPtr << " " << myInt;
   delete myPtr; // Deallocated from heap

   MyFct(); // Stack grows, then shrinks

   return 0;
}
```

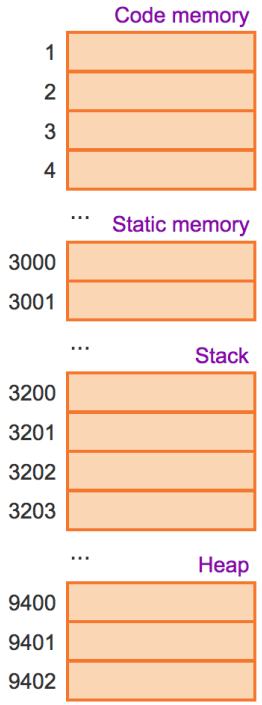


The code regions store program instructions. myGlobal is a global variable and is stored in the static memory region. Code and static regions last for the entire program execution.

Pointers

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

```
int* p;
```

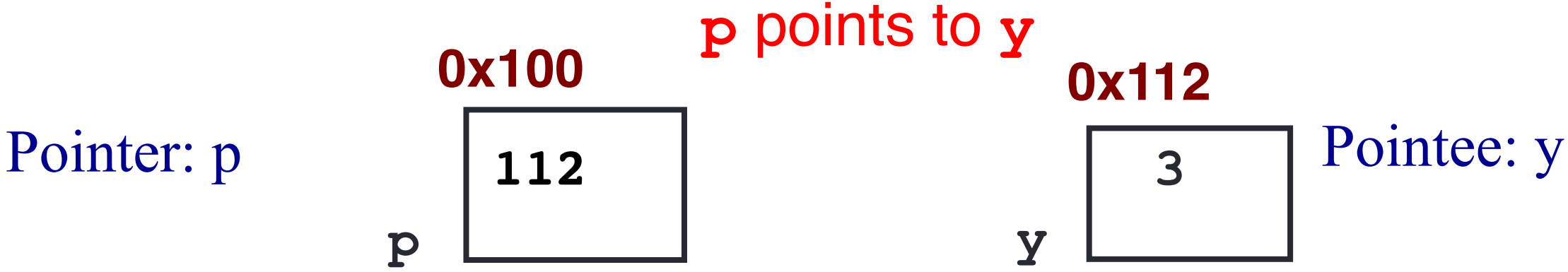


How to make a pointer point to something

```
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

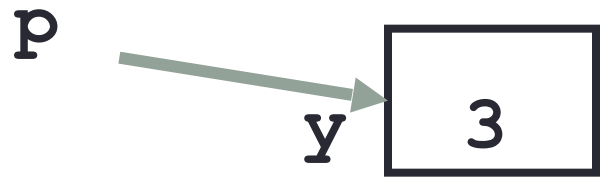


You can change the value of a variable using a pointer !

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

*p = 5;
```

Two ways of changing the value of a variable

- Change the value of y directly:



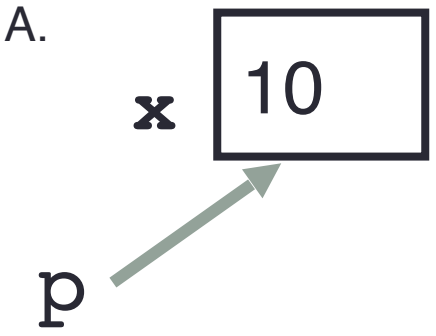
- Change the value of y indirectly (via pointer p):

Tracing code involving pointers

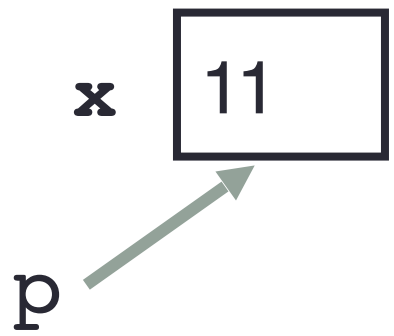
```
int* p;
int x = 10;
p = &x;
*p = *p + 1;
```

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

A.



B.



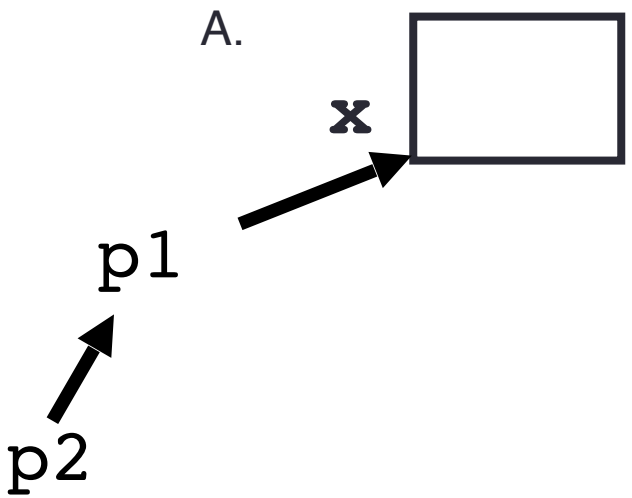
C. Neither, the code is incorrect

Pointer assignment

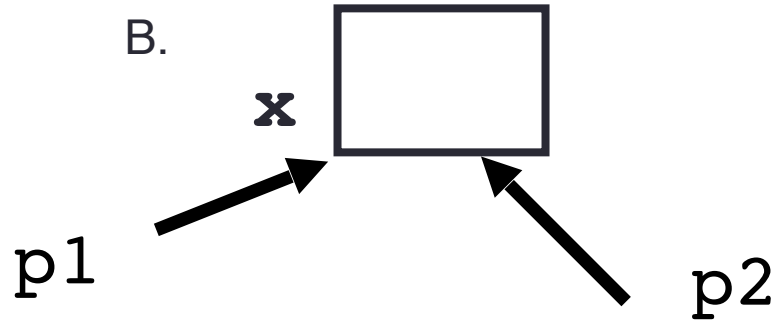
```
int* p1, *p2, x;
p1 = &x;
p2 = p1;
```

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

A.

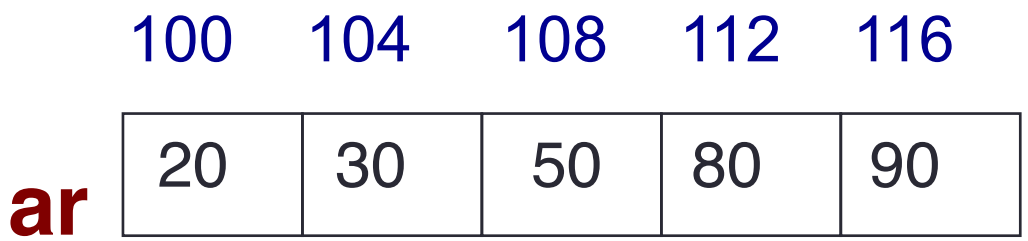


B.



C. Neither, the code is incorrect

Arrays and pointers



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

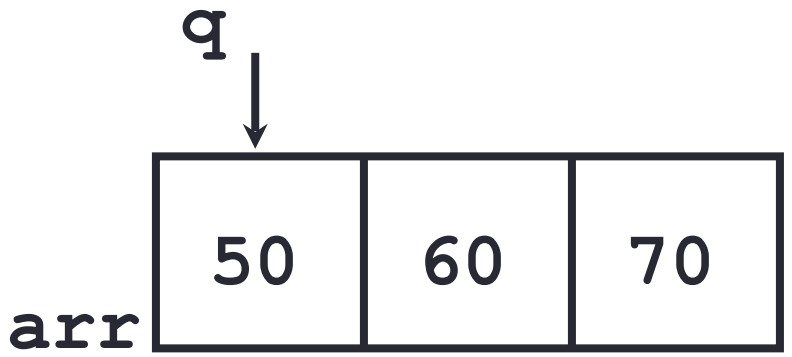
- Use pointers to pass arrays in functions
- Use *pointer arithmetic* to access arrays more conveniently

Pointer Arithmetic

```
int ar[]={20, 30, 50, 80, 90};
int* p;
p = arr;
p = p + 1;
*p = *p + 1;
```

Draw the array ar after the above code is executed

```
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?

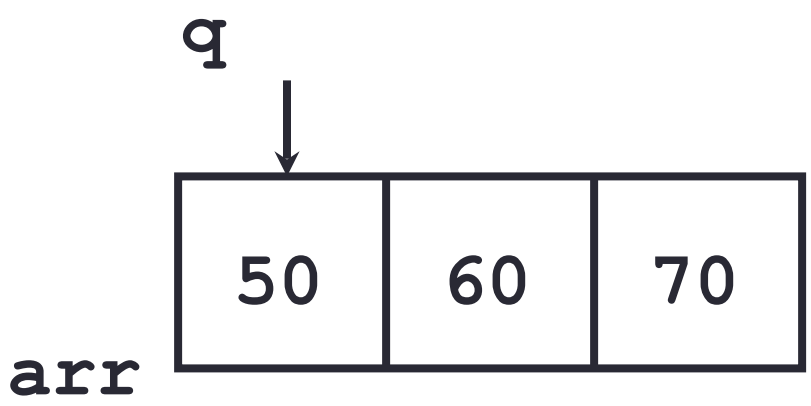
```
void IncrementPtr(int** p){
    p++;
}

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.

References in C++

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

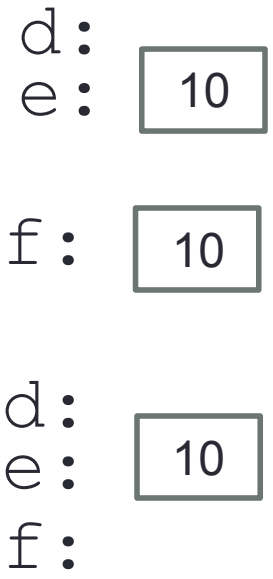
A reference in C++ is an alias for another variable

References in C++

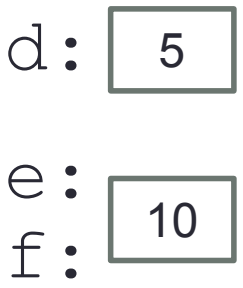
```
int main() {
   int d = 5;
   int &e = d;
   int f = 10;
   e = f;

}
```

How does the diagram change with this code?

A.



B.



C.

D. Other or error

Passing arguments to functions by reference

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

void ConvHrMin(int timeVal, int& hrVal, int& minVal) {
   hrVal = timeVal / 60;
   minVal = timeVal % 60;
}

int main() {
   int totTime;
   int usrHr;
   int usrMin;

   totTime = 0;
   usrHr = 0;
   usrMin = 0;

   cout << "Enter total minutes: ";
   cin >> totTime;

   ConvHrMin(totTime, usrHr, usrMin);

   cout << "Equals: ";
   cout << usrHr << " hrs ";
   cout << usrMin << " min" << endl;

   return 0;
}
```

Suppose the user enters a value of 125 for totTime
What is the output of the code?

Next time

- Dynamic Memory Management in C++