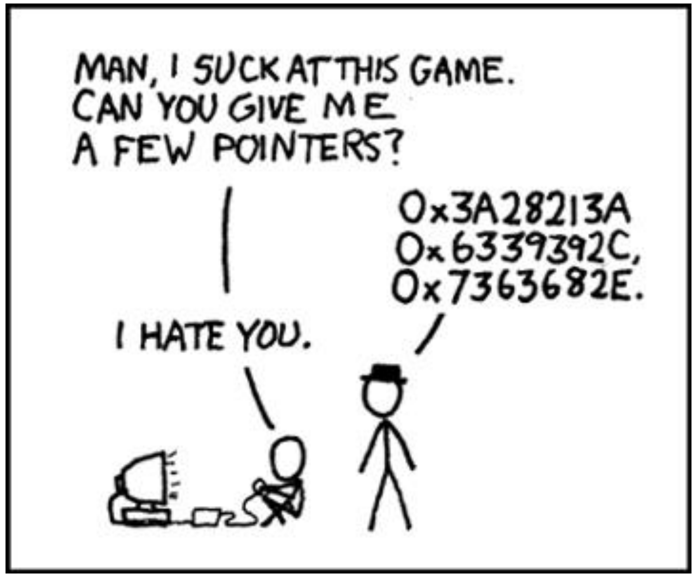
ARRAYS AND POINTERS



Problem Solving with Computers-I

https://ucsb-cs16-wi17.github.io/

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

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

GitHub

Are code A and code B equivalent?

[✓ A. Yes]
B. No

Code A

```
int sc[5]={65,85,97,75,95};
double sum=0;
   for (int i=0; i<5; i++){
       sum+=sc[i];
   }
double avg=sum/5;
```

Code B

```
int sc[5]={65,85,97,75,95};
double sum=0;
   for (int i : sc){
       sum+=i;
   }
double avg=sum/5;
```

$\rightarrow$ Range-based for loop

i gets the value of the next element of the array sc in every iteration

Passing arrays as arguments to functions

Write all possible valid declarations of a function that takes an integer array of scores as parameter and returns the average of the scores

```
double    getAverage (int arr[ ], int len)
double    getAverage ( int *arr, int len);

int  sc[ ] = {65, 75, 85, 95, 100};
double avg = getAverage (sc, len);
```

This code works!

```
double getAverage(int sc[], int len){
    double sum=0;
    for (int i=0; i<len; i++){
        sum+=sc[i];
    }
    return (sum/len);
}
```

This code results in a compile time error

-Why?

Because arrays degenerate to simple pointers when they are passed to a function.

But what is a pointer anyway......

```
double getAverage_c11(int sc[], int len){
    double sum=0;
    for (int value:sc){
        sum+=value;
    }

    return (sum/len);
}
```

Pointers

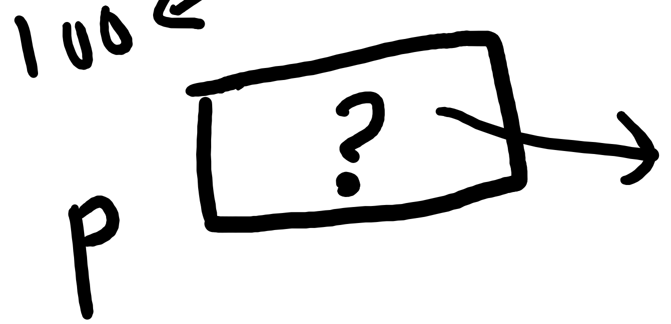
- **Pointer:** A variable that contains the address of another variable
- Declaration: *type* * pointer_name;

```
int *p;
```

This part specifies that p will hold the address of an int

'*' in declaration tells the compiler that p is a pointer

location of the pointer



value of the pointer - should be the address of an integer but its unknown right now!

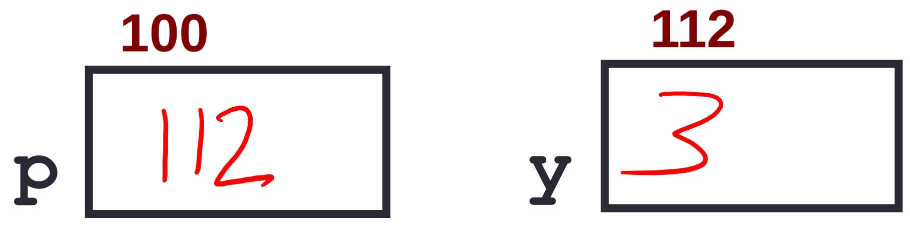
How do we initialize a pointer? int *p = NULL;

How to make a pointer point to something

```
int *p;
int y;
```

y = 3;

p = &y;



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

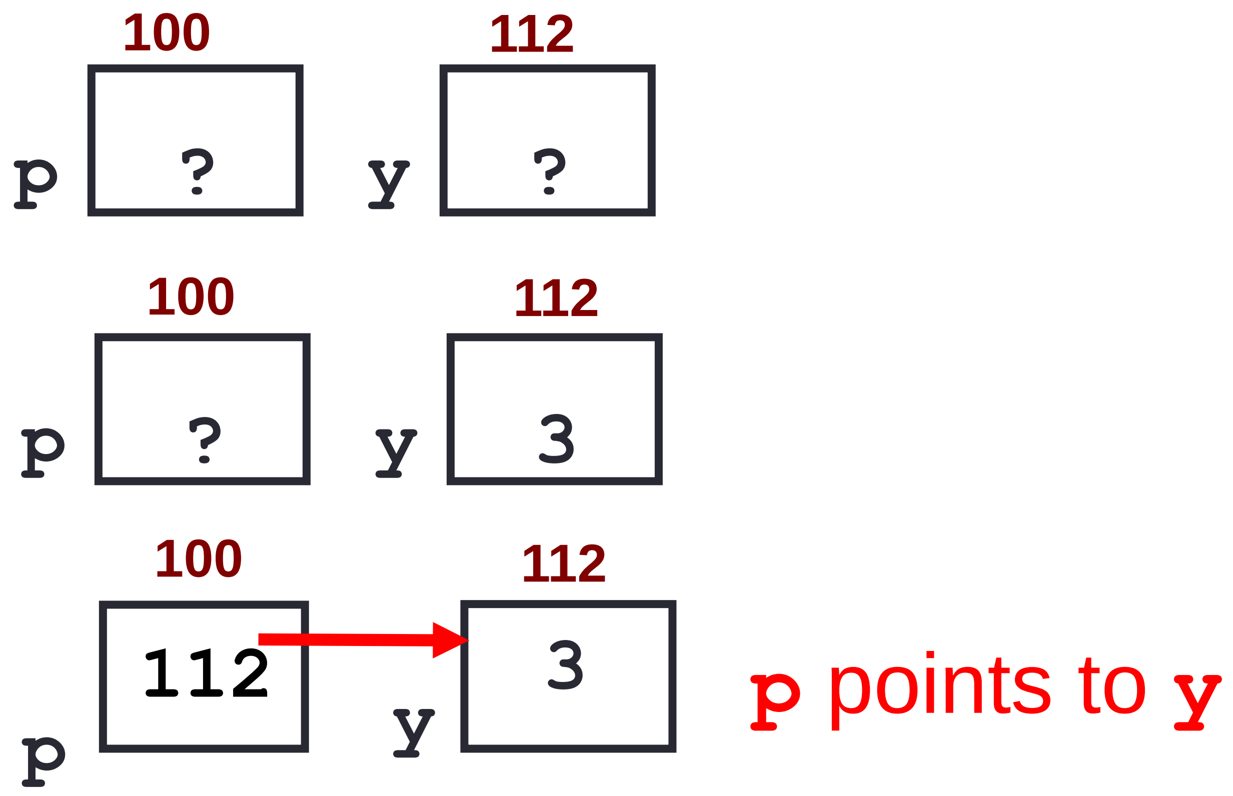
// p is now pointing to y

How to make a pointer point to something

```
int *p, y;
```

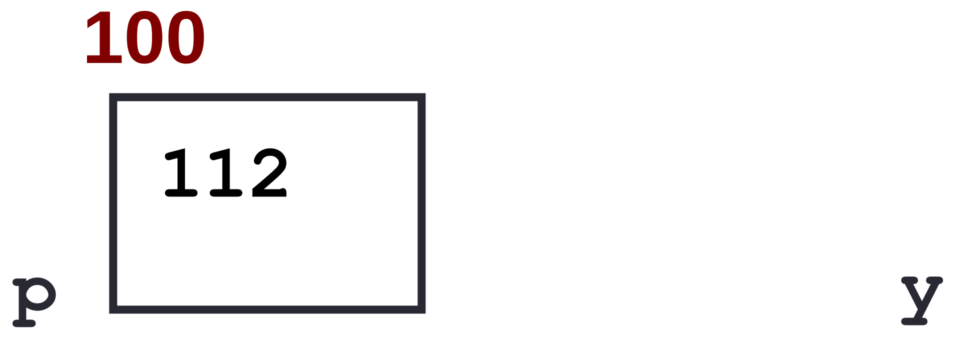
```
y=3;
```

```
p = &y;
```



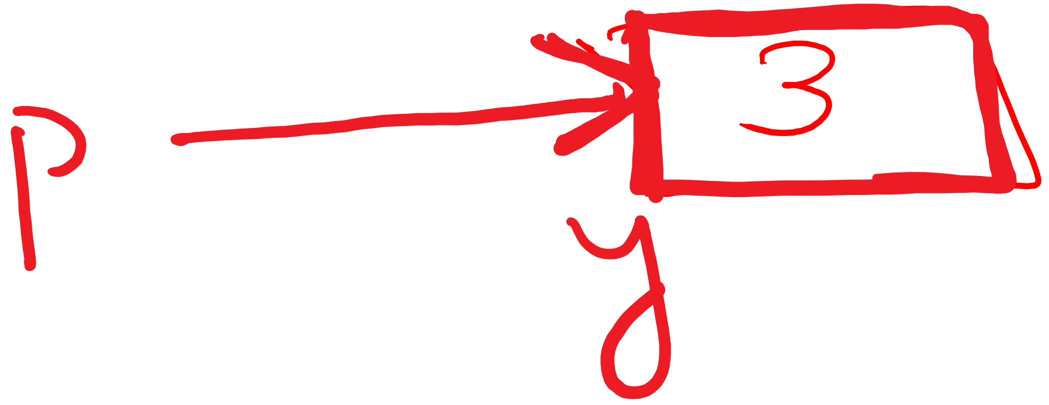
Pointer Diagrams:
Show relationship between pointers and pointees

Pointer: p



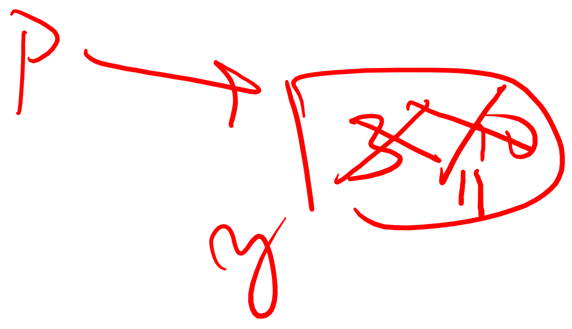
Pointee: y

This figure is a short-hand for the figure on the top



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

```
int *p, y;
y = 3;
p = &y;        // p is pointing to y
*p = 10;       // change the value of y through p
*p = *p + 1 ;  //get the value of y, add 1 and store it back in y
```



Use dereference * operator to left of pointer name

Tracing code involving pointers

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

$\rightarrow$ Same as

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

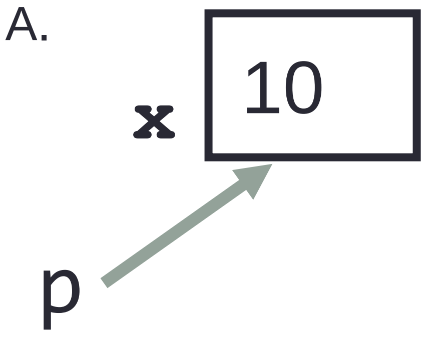
```
int *p;

char *p;
```

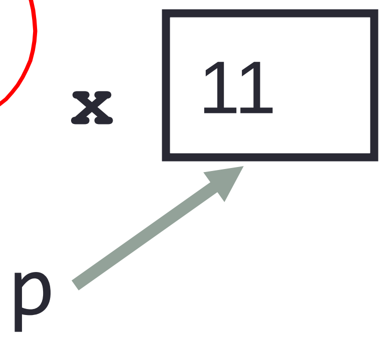
p is 4 bytes in both declarations

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

A.

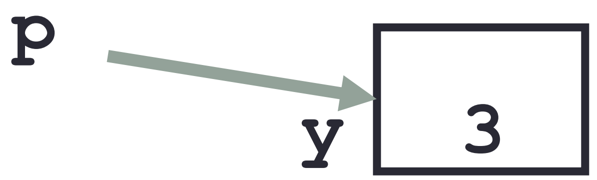


[✓ B.]



C. Neither, the code is incorrect

Two ways of changing the value of a variable



Change the value of y directly: y = 5;

Change the value of y indirectly (via pointer p): *p = 5;

Pointer assignment and pointer arithmetic: Trace the code

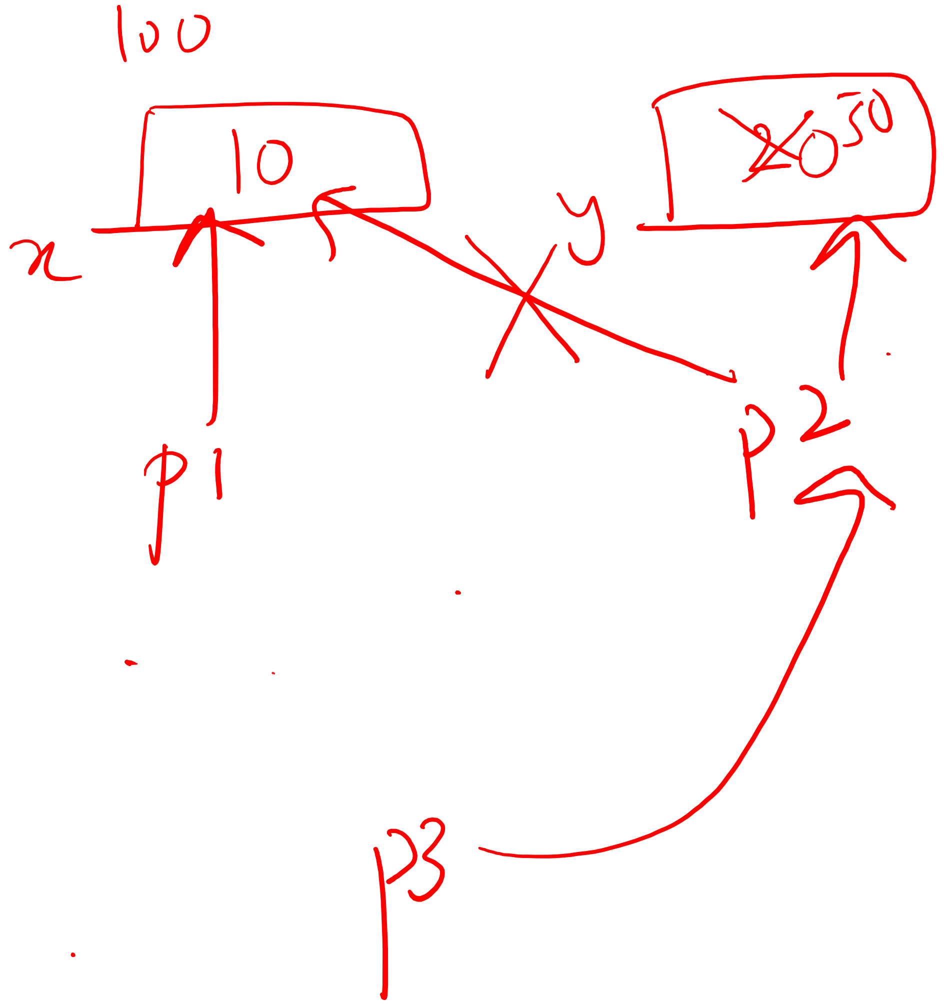
```
int x=10, y=20;

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

p2 = p1; //p2 points to the samething as p1

int **p3; // declare p3

p3 = &p2; // p3 points to p2
*p3 = &y; // p2 points to y
**p3 = 50; // changes the value of y to 50
```

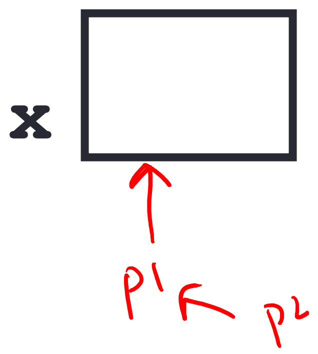


Pointer assignment

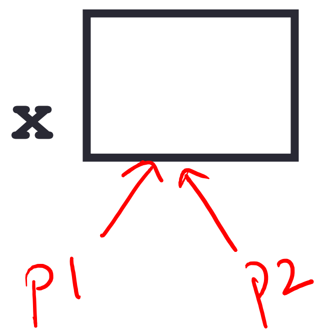
```
int *p1, *p2, x;
p1 = &x;
p2 = p1;
```

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

A.

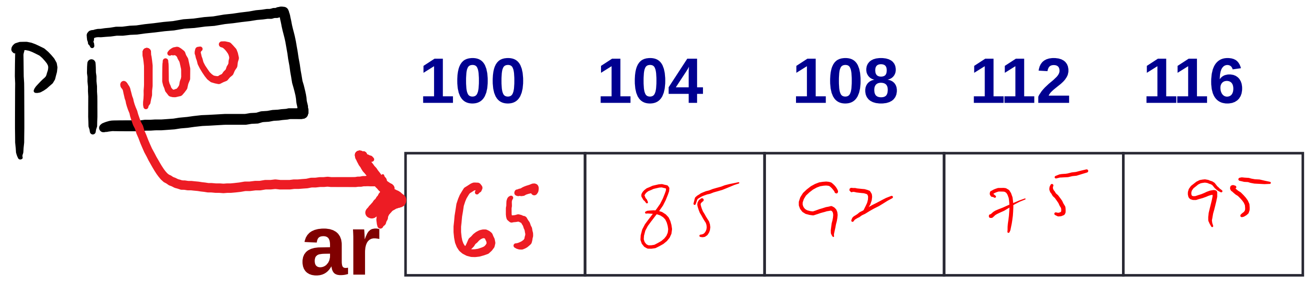


[✓ B.]



C. Neither, the code is incorrect

Arrays and pointers



(ar[0])
integer

- `ar` holds the address of the first element (like a pointer)
- `ar[0]` is the same as `*ar`
- Use pointers to pass arrays in functions

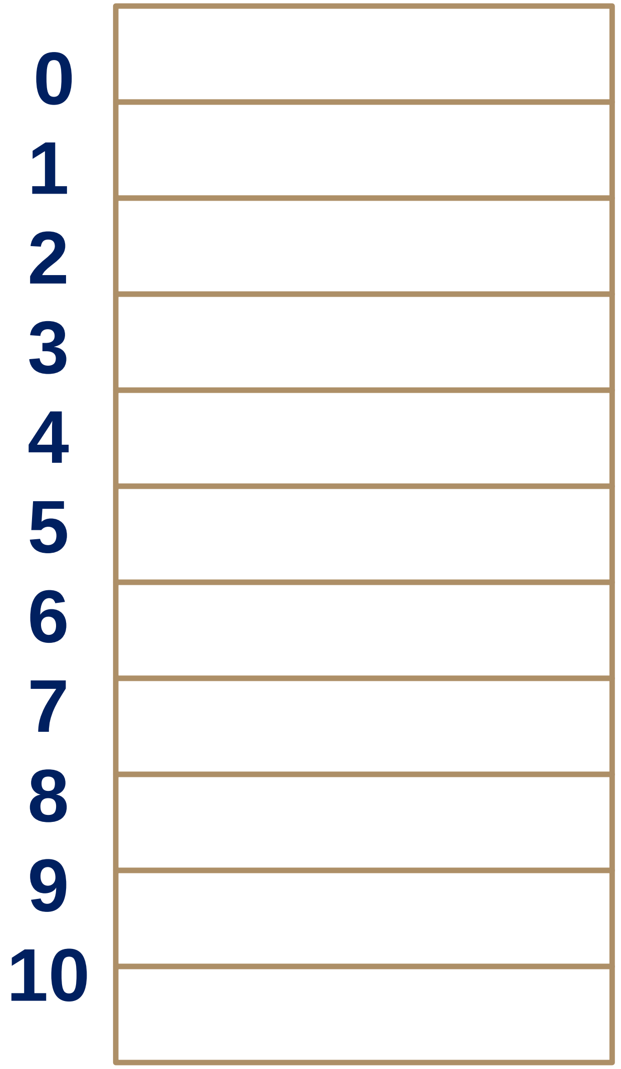
```
int ar[5]={65, 85, 97, 75, 95};
int *p;
```

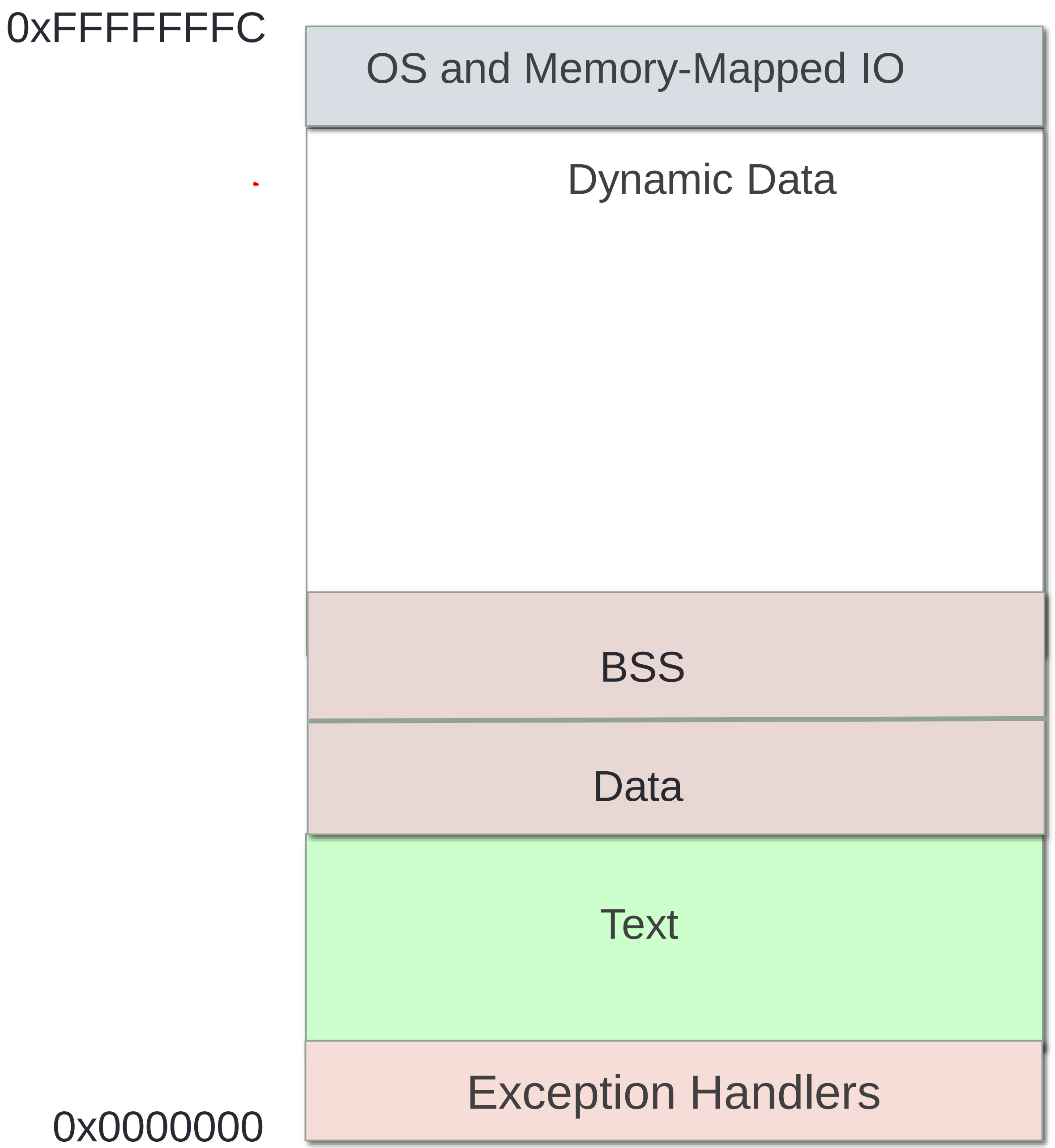
ar[0] = 10; same as $\equiv$ *ar = 10;

p = ar;

p[0] = 10; same as $\equiv$ *p = 10; same as $\equiv$ ar[0] = 10;

Your program in memory at runtime, runtime stack





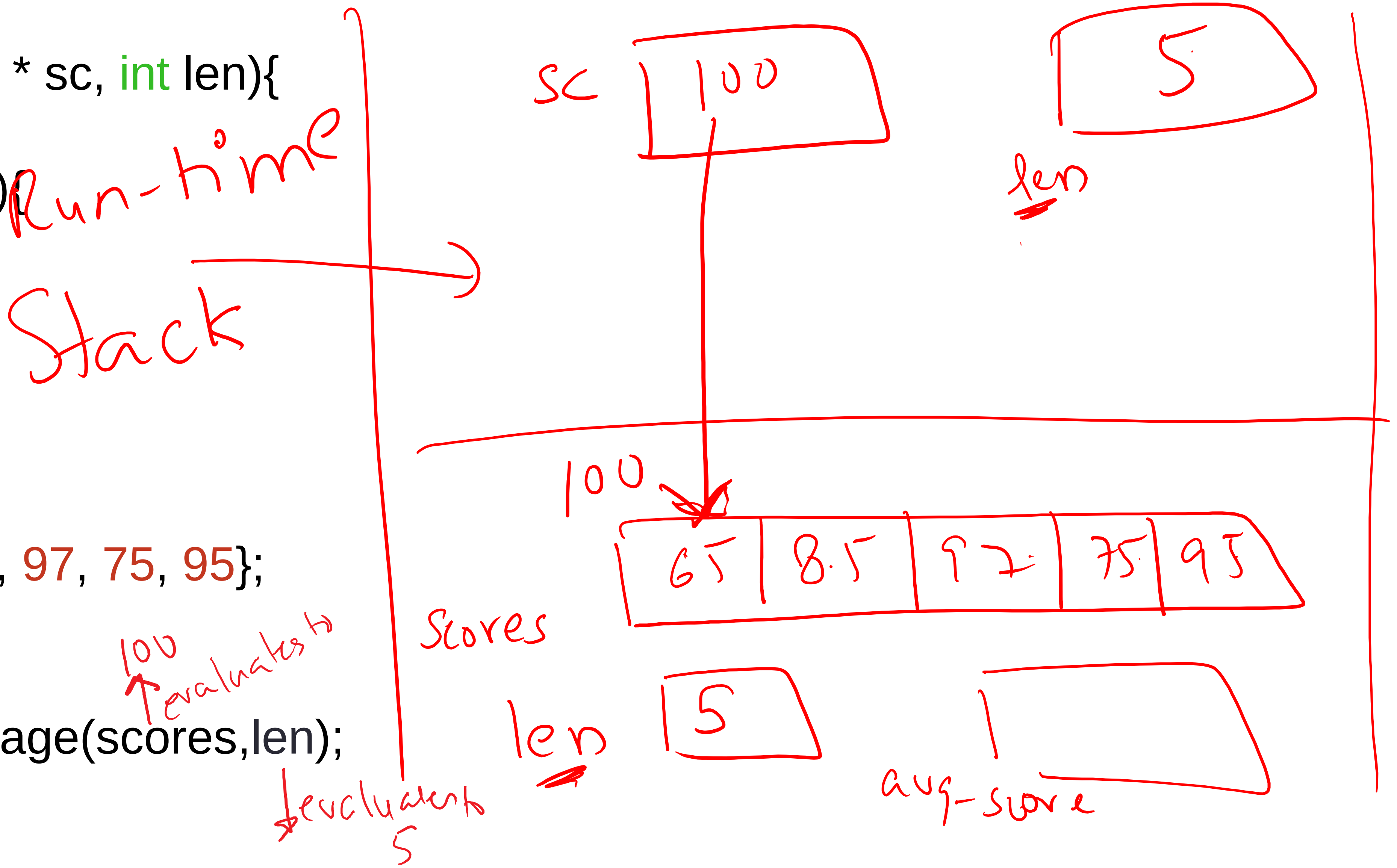
Mechanics of function calls on the run-time stack

```
double getAverage(int * sc, int len){
  double sum=0;
  for (int i=0; i<len; i++){
        sum+=sc[i];
  }
  return (sum/len);
}

int main(){
  int scores[5]={65, 85, 97, 75, 95};
  int len = 5
  double avg_score;
  avg_score = getAverage(scores,len);
  cout<< avg_score;
}
```

100 evaluates to

evaluates to 5



Complex declarations in C/C++

How do we decipher declarations of this sort?

```
int *(*arr)[];
```

Read

*	as "pointer to" (always on the left of identifier)
[]	as "array of" (always to the right of identifier)
( )	as "function returning" (always to the right …)

Ref: Rick Ord
http://ieng9.ucsd.edu/~cs30x/rt_lt.rule.html

Complex declarations in C/C++

Right-Left Rule

```
int *(*arr)[];
```

Illegal combinations include:

[]() - cannot have an array of functions
()() - cannot have a function that returns a function
()[] - cannot have a function that returns an array

Step 1: Find the identifier

Step 2: Look at the symbols to the right of the identifier. Continue right until you run out of symbols *OR* hit a *right* parenthesis ")"

Step 3: Look at the symbol to the left of the identifier. If it is not one of the symbols '*', '(), '[]' just say it. Otherwise, translate it into English using the table in the previous slide. Keep going left until you run out of symbols *OR* hit a *left* parenthesis "(".

Repeat steps 2 and 3 until you've formed your declaration.

Complex declarations in C/C++

```
int i;
int *i;
int  a[10];
int f( );
int **p;
int (*p)[];
int (*fp) ( );
int *p[];
int af[]( );
int *f();
int fa()[];
int ff()();
int (**ppa)[];
int (*apa[ ])[ ] ;
```

Next time

- What can go wrong when using pointers
- References
- Pointers and structs
- Mechanics of function calls contd.–call by reference