

ARRAYS AND POINTERS



Problem Solving with Computers-I

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

C++

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

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



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;
```

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

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?

```
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;
```

How do we initialize a pointer?

How to make a pointer **point to** something

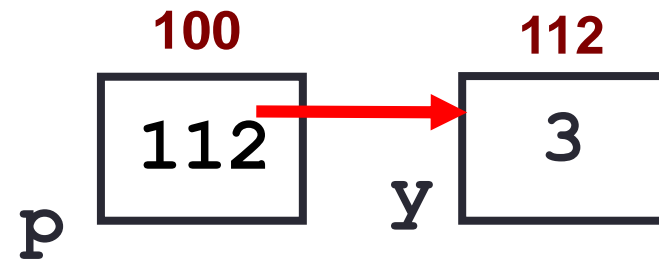
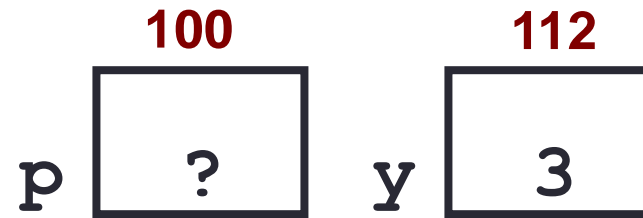
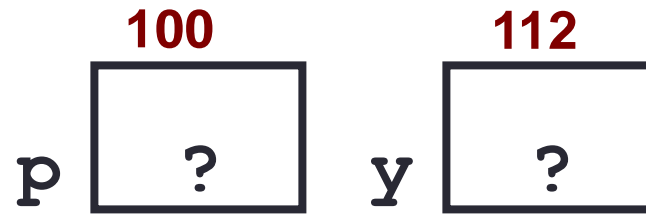
```
int *p;  
int y;
```



To access the location of a variable, use the address operator `'&'`

How to make a pointer **point to** something

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



p points to y

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;
```

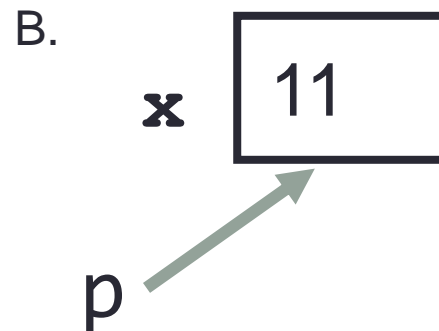
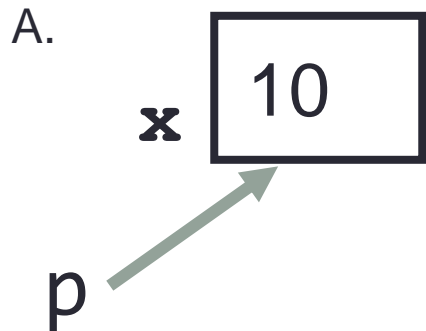
```
*p = 5;
```

Use dereference * operator to left of pointer name

Tracing code involving pointers

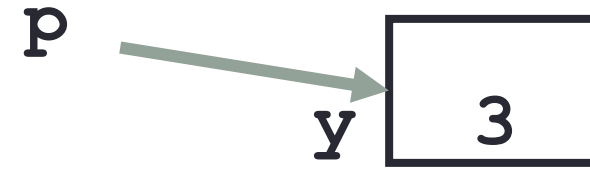
```
int *p, x=10;  
p = &x;  
*p = *p + 1;
```

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



C. Neither, the code is incorrect

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.

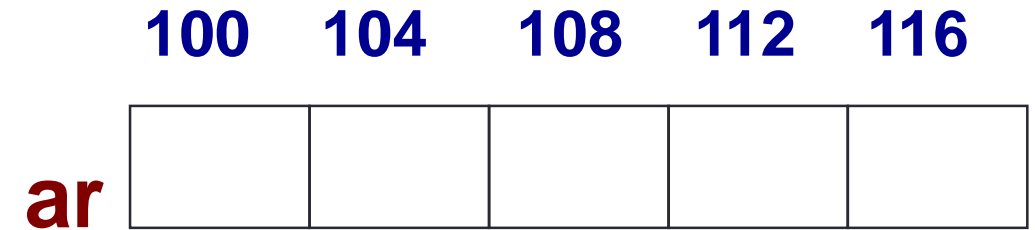


B.



C. Neither, the code is incorrect

Arrays and pointers

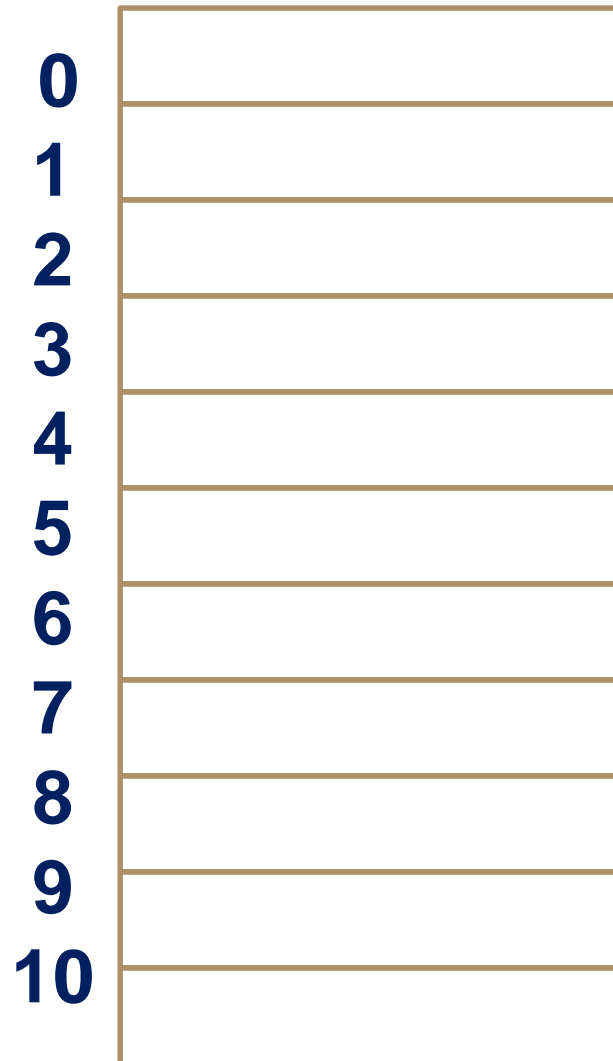


- `ar` holds the address of the first element (like a pointer)
- `ar[0]` is the same as `*ar`
- Use pointers to pass arrays in functions

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

```
int *p;
```

Your program in memory at runtime, runtime stack



0xFFFFFFFFFC

OS and Memory-Mapped IO

Dynamic Data

BSS

Data

Text

Exception Handlers

0x00000000

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;
}
```


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)[];
```

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.

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

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