STRUCTS REFERENCES POINTERS (REVIEW)

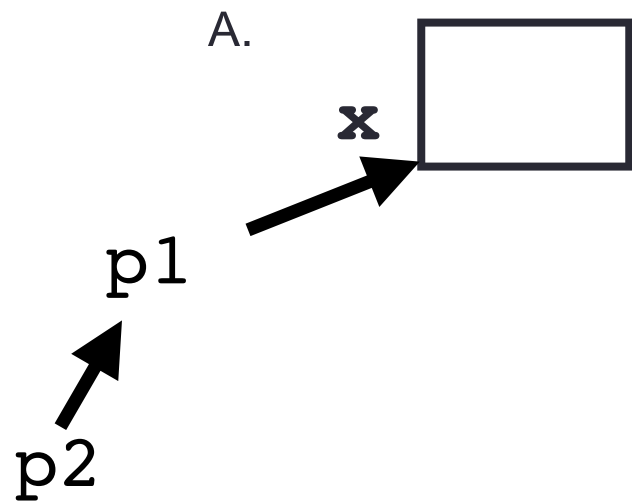
Problem Solving with Computers-I

Pointer assignment

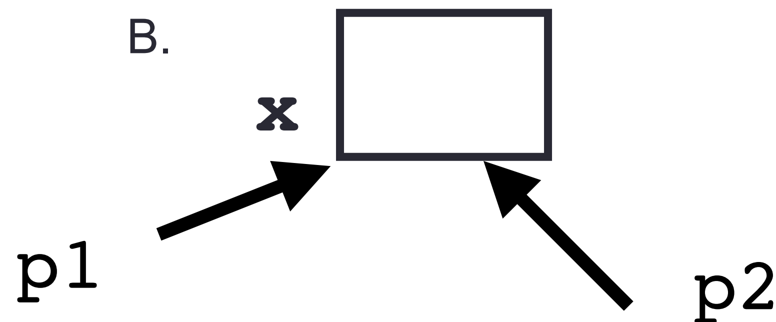
```
int *p1, *p2, x;
p1 = &x;
p2 = p1;
```

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

A.



B.



C. Neither, the code is incorrect

C++ structures (lab05)

A **struct** is a data structure composed of simpler data types.

```
struct Point {
    double x; //member variable of Point
    double y; //member variable of Point
};
```

Think of Point as a new data type

```
Point p1;                // Declare a variable of type Point
Point p1 = { 10, 20};   //Declare and initialize
```

C++ structures (lab05)

- A **struct** is a data structure composed of simpler data types.

```
struct Point {
    double x; //member variable of Point
    double y; //member variable of Point
};
```

- Access the member variables of p1 using the dot '.' operator

```
Point p1;
p1.x = 5;
p1.x = 10;
```

- Access via a pointer using the -> operator

```
Point* q = &p1;
(*q).x = 5;
(*q).x = 10;
```

Which of the following is/are incorrect statement(s) in C++?

```
struct Point {
    double x;
    double y;
};
```

```
struct Box {
    Point ul; // upper left corr
    double width;
    double height;
};
```

A. `ul.x = 10;`

B. `Box b1 = {{500, 800}, 10, 20};`

C. `Box b1, b2; b1.ul = {500, 800};`

D. A and C

E. None of the above are incorrect

Passing structs to functions

- Write a function that prints the x and y coordinates of a `Point`
- Write a function that takes a `Point` as parameter and initializes its x and y coordinates

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. d: e: [10] f: [10]

B. d: [5] e: f: [10]

C. d: e: f: [10]

D. Other or error

Pointers and references: Draw the diagram for this code

```
int a = 5;
int & b = a;
int* pt1 = &a;
```

What are three ways to change the value of ‘a’ to 42?

Call by reference: Modify to correctly swap a and b

```
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;
}
```

Passing structs to functions by reference

- Write a function that takes a `Point` as parameter and initializes its x and y coordinates

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)`

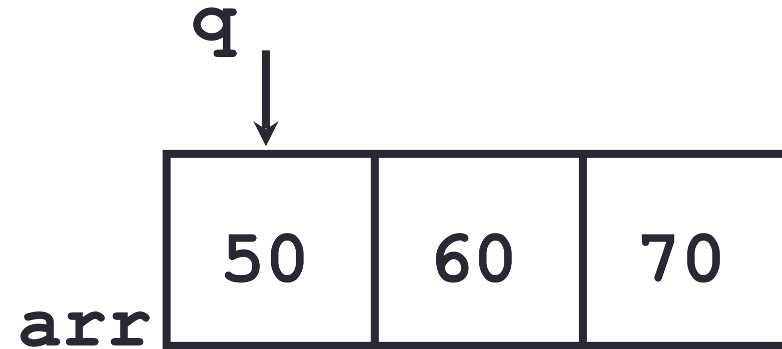
- Use pointers to pass arrays in functions
- Use *pointer arithmetic* to access arrays more conveniently

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

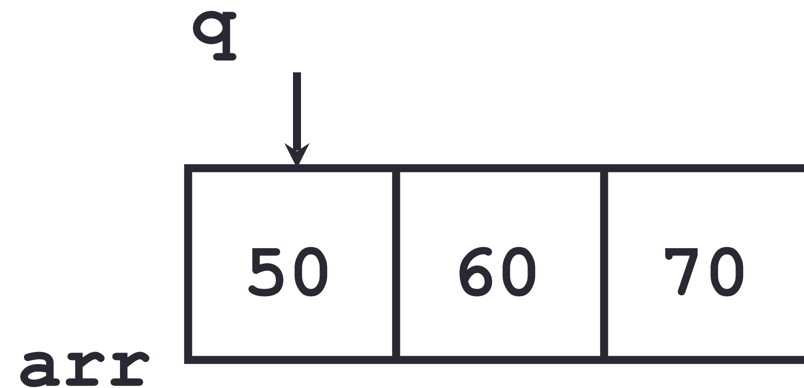
B. **'q'** points to the first element in the array with value 50

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

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 Arithmetic Question

How many of the following are invalid?

I. pointer + integer (ptr+1)
II. integer + pointer (1+ptr)
III. pointer + pointer (ptr + ptr)
IV. pointer – integer (ptr – 1)
V. integer – pointer (1 – ptr)
VI. pointer – pointer (ptr – ptr)
VII. compare pointer to pointer (ptr == ptr)
VIII. compare pointer to integer (1 == ptr)
IX. compare pointer to 0 (ptr == 0)
X. compare pointer to NULL (ptr == NULL)

#invalid
A: 1
B: 2
C: 3
D: 4
E: 5

Next time

- Arrays of structs
- Dynamic memory allocation