Review Questions

64-bit example
(pointers are 64-bits wide)

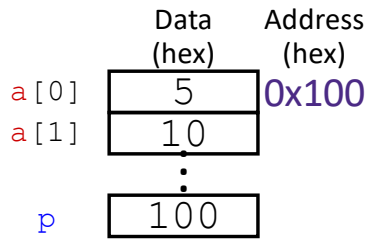
- ```
  int x = 351;
  char* p = &x;
  int ar[3];
  ```

- How much space does the variable p take up?
  - A. 1 byte
  - B. 2 bytes
  - C. 4 bytes
  - D. 8 bytes

- Which of the following expressions evaluate to an address?
  - A. `x + 10`
  - B. `p + 10`
  - C. `&x + 10`
  - D. `*(&p)`
  - E. `ar[1]`
  - F. `&ar[2]`

Question: The variable values after Line 3 executes are shown on the right. What are they after Line 5?

```
1  void main() {
2    int a[] = {0x5,0x10};
3    int* p = a;
4    p = p + 1;
5    *p = *p + 1;
6  }
```



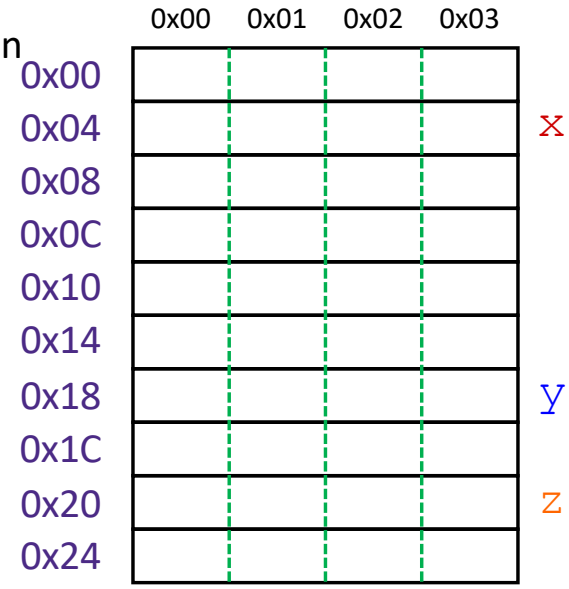
	p	a[0]	a[1]
(A)	0x101	0x5	0x11
(B)	0x104	0x5	0x11
(C)	0x101	0x6	0x10
(D)	0x104	0x6	0x10
```

Assignment in C - Handout

32-bit example
(pointers are 32-bits wide)

& = "address of"
* = "dereference"

- left-hand side = right-hand side;
  - LHS must evaluate to a *location*
  - RHS must evaluate to a *value*
  - Store RHS value at LHS location
- `int x, y;`
- `x = 0;`
- `y = 0x3CD02700;`
- `x = y + 3;`
- `int* z = &y + 3;`
- `*z = y;`



Arrays in C - Handout

Arrays are adjacent locations in memory storing the same type of data object

`a` (array name) returns the array's address

`&a[i]` is the address of `a[0]` plus `i` times the element size in bytes

Declaration: `int a[6];`

Indexing:
```
a[0] = 0x015f;
a[5] = a[0];
```

No bounds checking:
```
a[6] = 0xBAD;
a[-1] = 0xBAD;
```

Pointers: `int* p;`

equivalent:
```
p = a;
p = &a[0];
```

```
*p = 0xA;
```

array indexing = address arithmetic
(both scaled by the size of the type)

equivalent:
```
p[1] = 0xB;
*(p+1) = 0xB;
```

```
p = p + 2;
```

```
*p = a[1] + 1;
```