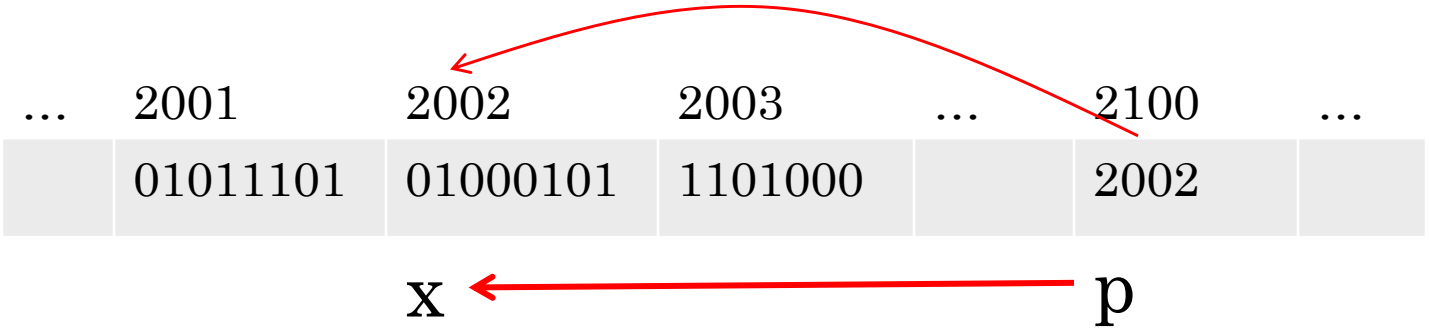
CSE 220 – C Programming

Pointers

Pointer Variables



- In memory: each byte has an address
- Pointers: used to store the address
- Store the address of x in pointer variable p:
  p points to x

Pointer Declaration

- Add a * before the variable name to indicate a pointer variable

```
int *p;          //p can only point to an integer
double *q;       //q can only point to a double
char *r;         //r can only point to a character
```

- Pointer variables can appear with other declarations:

```
int a, *b, c[10], *q;
int * b, a;
```

a in an int
b is a pointer to an int

Address Operator: &

```
int i, *p;
```

- p does not point to any particular place
- Must initialize p before using it

```
p = &i;
```

- Assign address of i to p using address operator &
- p is a pointer to i
- *p is the value that p points to

Declare and initialize in one step (optional material):

```
int i, *p = &i; //i must be declared first
```

- Combine declaration and initialization

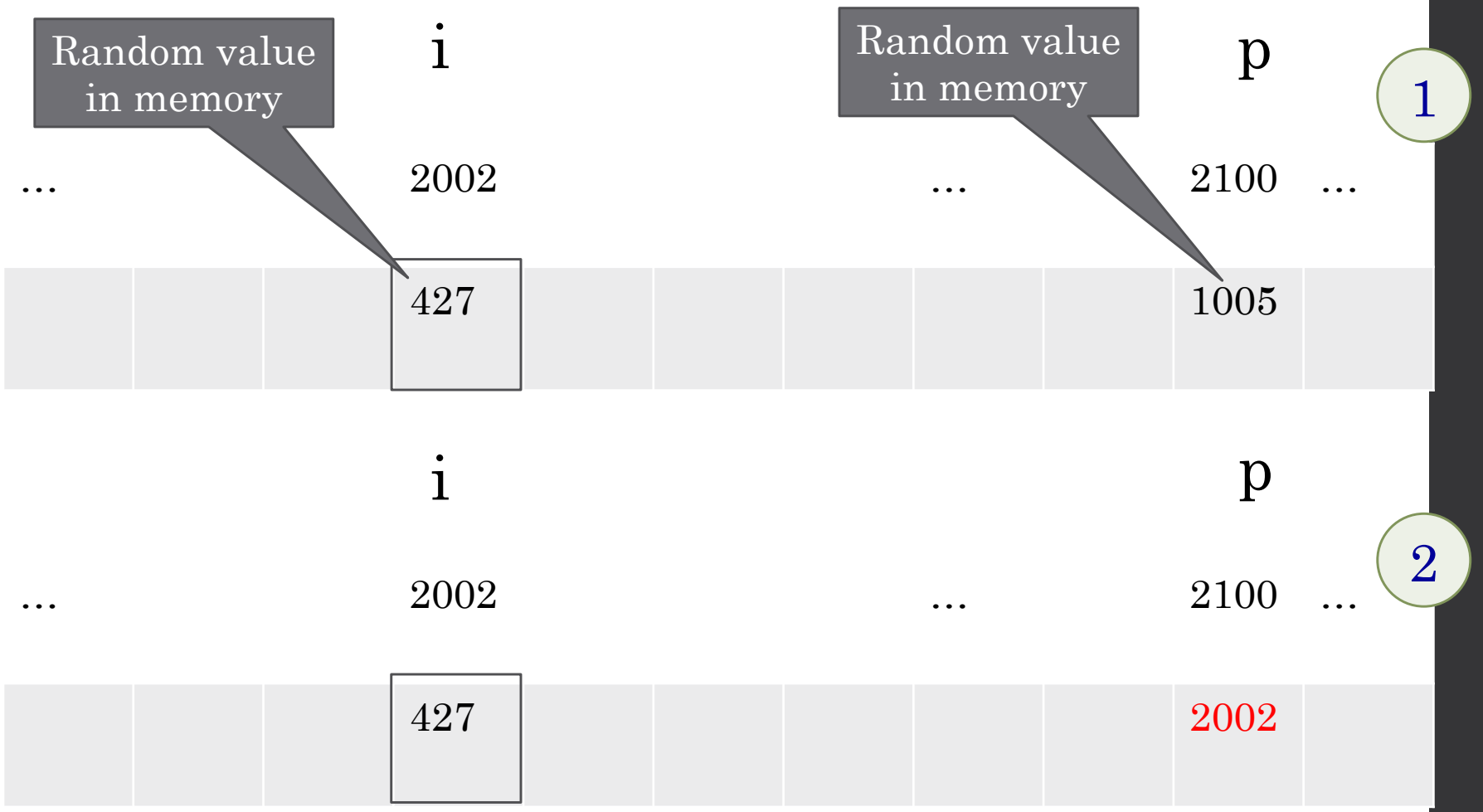
Which of these variables are pointers?

```
int x, *y;
int *z;
char *a;
x = 4; y = &x; z = y;
```

loc.	124	125	126	127	128	129	130
value	4	124	?	?	?	124	?
	x	z	a			y	

- x
- y
- z
- a

```
int i, *p;
p = &i; //address of i: 2002
```



Indirection Operator: *

- The indirection operator * is used to access the value pointed to by a pointer:
  - Must initialize p before using it

```
int i = 11;

int *p = &i;

printf("%d\n", *p);    //prints 11

printf("%p\n", p);     // 0x7fffffffda6c
```

Examples

```
int i, j, *p;
j = *&i;
```

&i: pointer to i
*(&i): value pointed to by the pointer
*&i same as i

```
i = 3;
int *p = &i;
i = 4;
printf("%d\n", *p);
```

p points to i, so it prints the current value of i

Examples

```
int i = 3, *p = &i;
*p = 4;
printf("%d\n", i);
printf("%d\n", *p);
```

Line 2: Assigns 4 to the value pointed to by p, so assigns 4 to i
Line 3: Prints 4
Line 4: Also, prints 4

```
int *q;
printf("%d\n", *q);
*q = 1;
```

q is uninitialized, does not point to any particular location
Undefined behavior

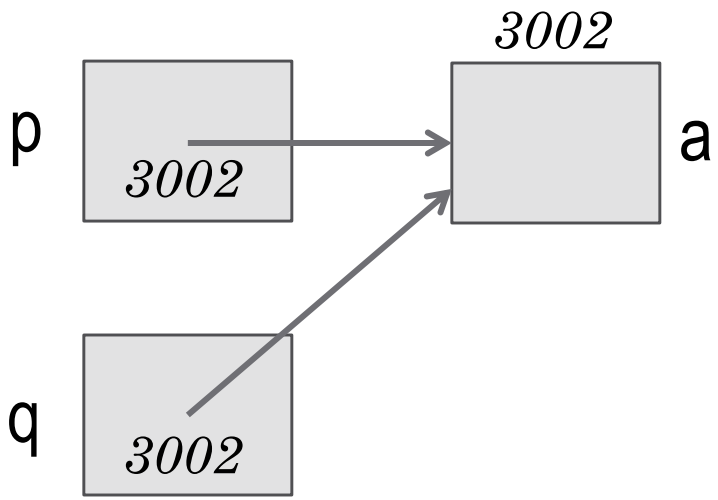
What does this code output?

```
int a, *b;
int c, *d;
a = 4; c = 5;
b = &a; d = &c;
printf("%d%d%d%d", a, *b, c, *d);
```

- Error
- 4545
- 4455
- I don't know

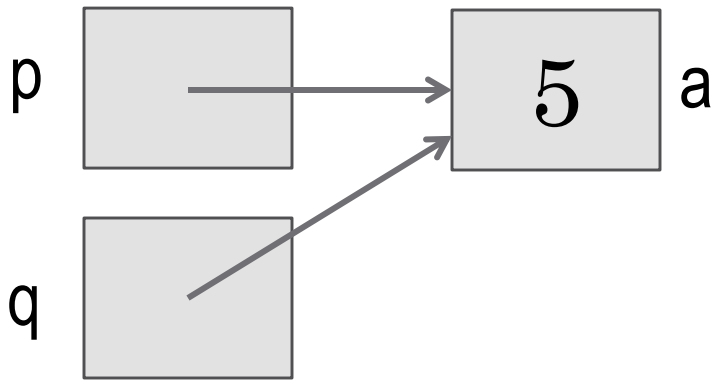
Pointer Assignment

```
int a, b, *p, *q;
p = &a;     //Copy the address of a into p
q = p;      //Copy the content of p into q
```



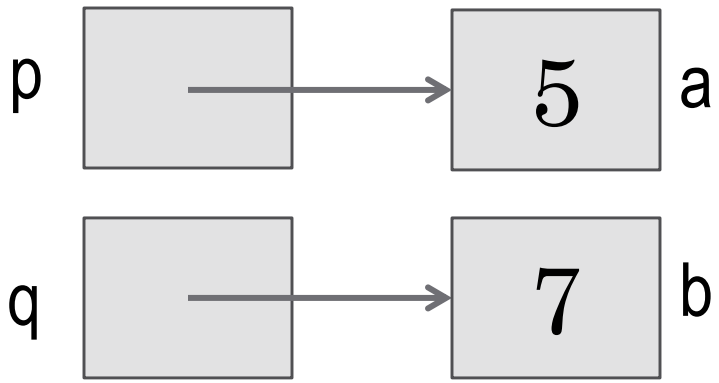
Pointer Assignment

```
*p = 5;
//Change the value pointed to by p to 5
printf("%d\n", *q);
```

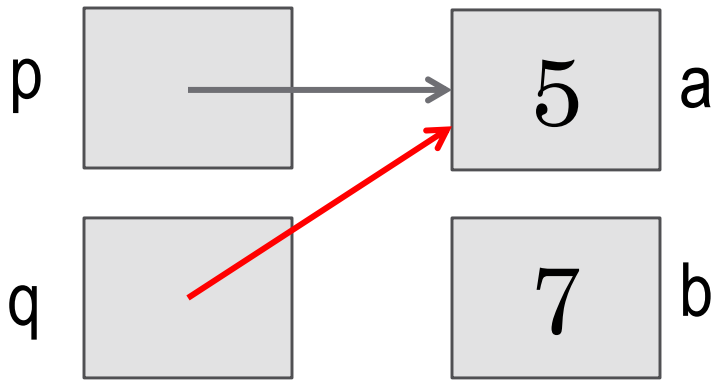


Pointer Assignment

```
int a = 5, *p = &a;

int b = 7, *q = &b;
```



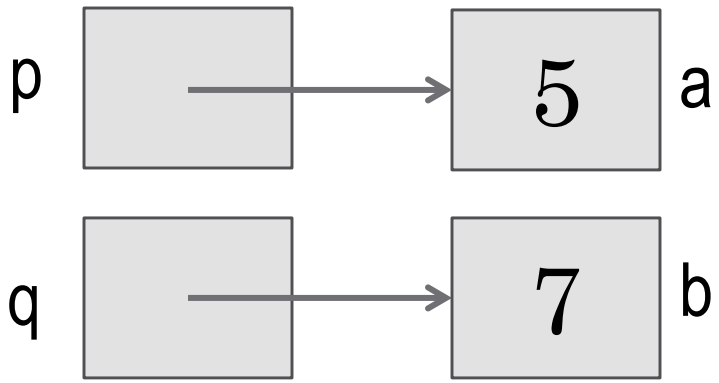

```
q = p;
```



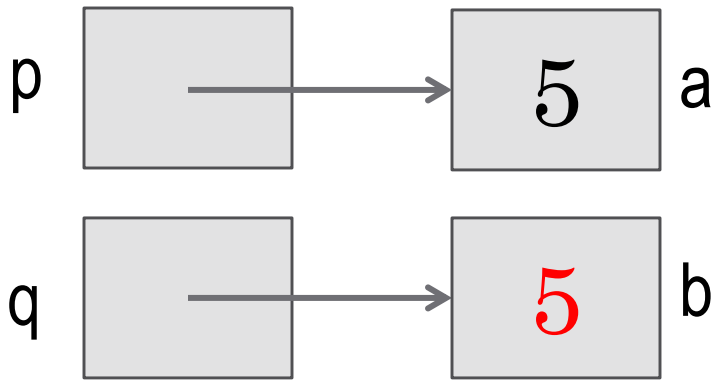
Copies the content of p to the content of q

Pointer Assignment

```
int a = 5, *p = &a;

int b = 7, *q = &b;
```




```
*q = *p;
```



Copies value pointed to by p to value pointed to by q

What does this code output?

```
int a, *b, c, *d;
a = 4; c = 5;
b = &a; d = &c;
*d = 6;
printf("%d%d%d%d", a, *b, c, *d);
```

- 4455
- 4466
- 4465
- I don't know

Pointers as Arguments

- C passes arguments by value
- What if we want to modify the value?

```
void triple(int a) {
    a = 3*a;
}
…
int a = 7;
triple(a);
printf("a = %d\n", a);
```

a = 7

```
int triple(int a) {
    return 3*a;
}
…
int a = 7;
a = triple(a);
printf("a = %d\n", a);
```

a = 21

Pointers as Arguments

- What if we want the function to modify multiple values?
- Solution: pass a pointer to the value

Pointers as Arguments

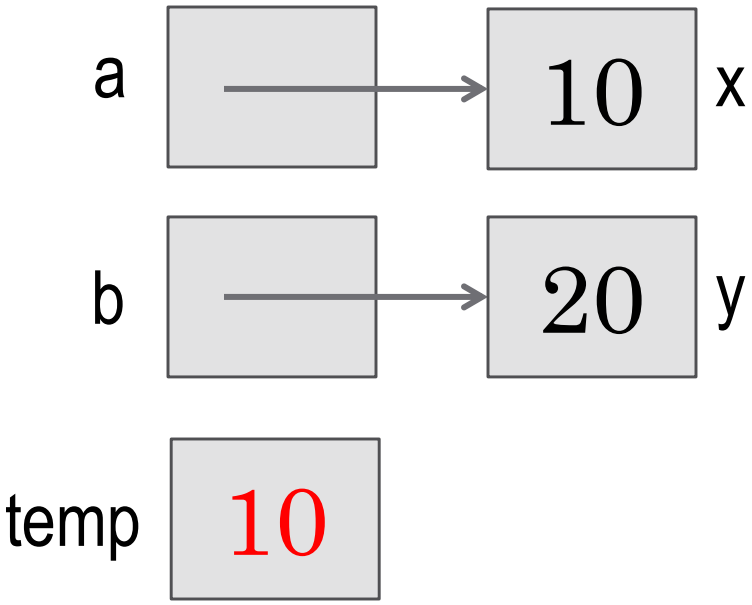
```
void swap(int a, int b) {
     int temp = a;
     a = b;
     b = temp;
}
int x = 10, y = 20;
swap(x, y);
printf("x = %d, y = %d",
      x, y);
```

x = 10, y = 20

After calling the function swap, the values of x and y remain the same.

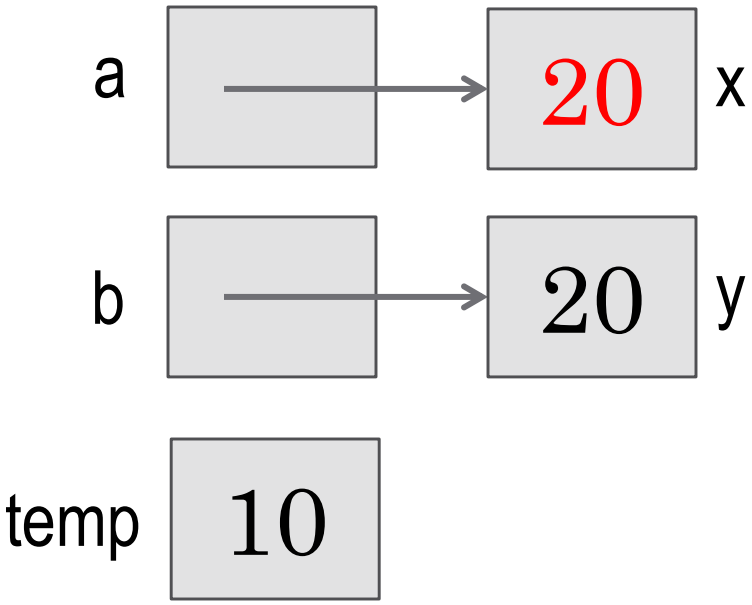
Pointers as Arguments

```
void swap(int *a, int *b)
{
     int temp = *a;
     *a = *b;
     *b = temp;
}

int x = 10, y = 20;
swap(&x, &y);
/* swap expects a pointer:
so pass &x instead of x,
&y instead of y */
```



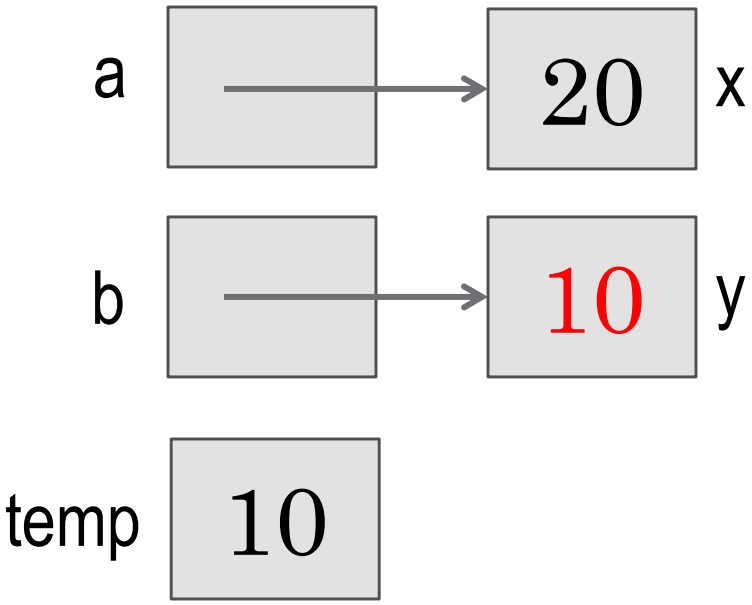
Pointers as Arguments

```
void swap(int *a, int *b)
{
     int temp = *a;
     *a = *b;
     *b = temp;
}

int x = 10, y = 20;
swap(&x, &y);
/* swap expects a pointer:
so pass &x instead of x,
&y instead of y */
```



Pointers as Arguments

```
void swap(int *a, int *b)
{
     int temp = *a;
     *a = *b;
     *b = temp;
}

int x = 10, y = 20;
swap(&x, &y);
/* swap expects a pointer:
so pass &x instead of x,
&y instead of y */
```



What does this code output?

```
void increment(int * x) {
    (*x)++;
}
...
int a = 4;
increment(&a);
printf("%d", a);
```

- 5
- Error
- 4
- I don't know

Scanf revisited

```
int x;

scanf(“%d”, &x);
```

Scanf expects a pointer, so the address of x is passed.

x is passed by reference, the change to x persists after the function returns.

Pointers as return values

- C allows functions to return a pointer

```
int *max(int *a, int *b) {
    if (*a > *b)
        return a;
    else
        return b;
}
int *p, i, j;
…
p = max(&i, &j);
```

Pitfalls

- Apply indirection to uninitialized pointer

```
int *p;
int x = *p;
//p pointing to a random location
```

- Fail to pass a pointer to a function when a pointer is expected

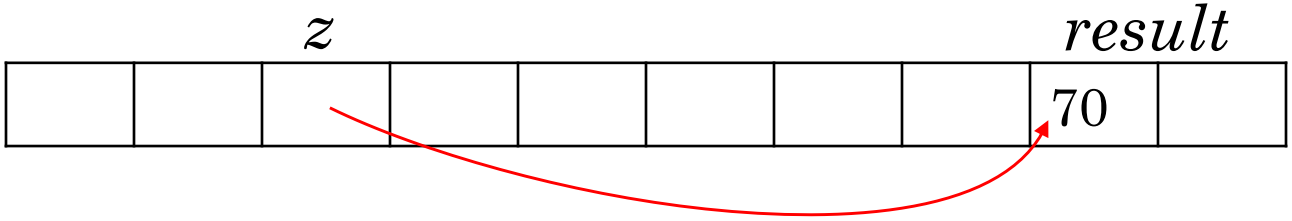
```
void multiply(int a, int b, int *result);
…
int x = 10, y = 20, z;
multiply(x, y, &z);
```

Pitfalls

- Return a pointer to a function variable

```
int * multiply(int a, int b) {
      int result = a*b;
      return &result;
}
…
int *z;
z = multiply(7, 10);
```

This location is destroyed after the function concludes.



What does this code output?

```
void double(int * x, int * y) {
    *y = *x * 2;
}
...
int a = 4, b;
increment(&a, b);
printf("%d", b);
```

- 5
- Error
- 4
- I don't know