APS 105 Lecture 12

Last lecture: Function & passing information into functions "call by value"

Today: More on getting information from functions + pointers

How to get information from a function?

① Using return statement (get only 1 output)

e.g. Write a C function that checks if an integer is a perfect square, like 1, 4, 9, 25, 36 ...
return is either true or false
e.g. sqrt(4) = 2, sqrt(5) = 2.236 ...
so if there is a difference between

```
bool isPerfectSquare( int x ){

    if( (int) sqrt(x) != sqrt(x))
        return false;
    else
        return true;
}
```

Multiple returns is bad, as if you have several if-else statements, you may forget to return in a certain condition.

```
bool isPerfectSquare(int x){
    return (int)sqrt(x) == sqrt(x);
}
```

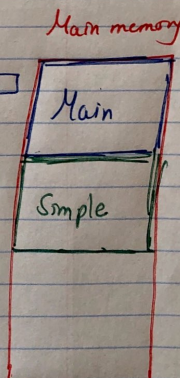
Simple & neat

(2) Using pointers as one of the parameters - complex, but powerful

Recall "call by value", whenever a parameter is passed to a function, a copy of the value is sent, not the variable itself.

E.g.

```
void simple(double P){
    P = P/2;
}
int main(void){
    double x = 5;
    simple(x);
    return 0;
}
```



→ x remains 5 since only a copy of x is passed to the function simple, not the variable itself

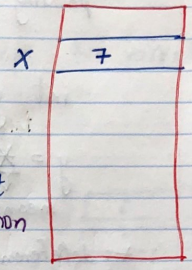
What if I want to change the value of x in simple?

POINTERS

What are pointers?

Main memory

```
int x = 7;
```



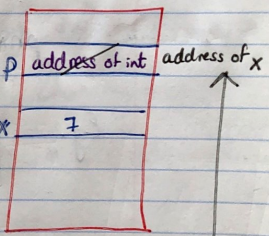
x is an int variable storing 7.

Pointers are also variables, but they store an address/location of another variable.

Declare a pointer

```
int *p;
```

→ this says that p will contain an address of an int variable



The data type of p is (int *).
You can also write
int* p;

① `int x = 7;` → declares variable x and set it to 7

② `p = &x;` → sets p to the address of x; we say p points to x

(& means "address of")

recall its usage in scanf

E.g.

	32-bit Address	Content of Address
	10	
p	14	18 ②
x	18	7 ①

I want to access the int variable that p is pointing to → Dereference a pointer

```
int y;
y = *p;   // set y to the contents of the address p: which is x
```

means value at address

*: dereference operator

as if we did y = x;

x	7
p	address of x
y	7

printf("%d \n", *p); → prints 7

E.g.

```
int main (void) {
    int x = 3, y = 4;
    int *a, *b;
    a = &x;
    b = &y;
    *a = 6;
    *b = *b + 3;
}
```

x	~~3~~ 6
y	~~4~~ 4+3 = 7
a	address of x
b	address of y

The value held at address of x is changed to 6

→ This changes the value of x, but **NOT** a. a still points to x. It still holds address of x

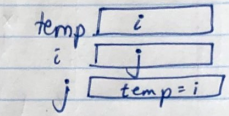
→ This changes the value of y, but **NOT** b. b still points to y. It still holds the address of y.

Use case of pointers

Write a C function that swaps two variables.

WRONG!

```
void swap (int i, int j) {
    int temp = i;
    i = j;
    j = temp;
}
```



i and j are swapped in the function swap, but since i and j were passed "by value" changing them in swap will **NOT** change them in main ☹

How about we do this:

```
void swap (int *pi, int *pj){
    ① int temp = *pi;
    ② *pi = *pj;
       *pj = temp;
}

int main (void){
    int i = 5, j = 7;
    swap (&i, &j);
    printf("i: %d, j: %d \n", i, j);
      // prints "i: 7, j: 5"
}
```

// *pi is i, *pj is j