ELEC 278

Tutorial Week 1 (?!)

2022 Fall

Instructors:
Dr. Jianbing Ni & Dr. Mohammadali Hedayati

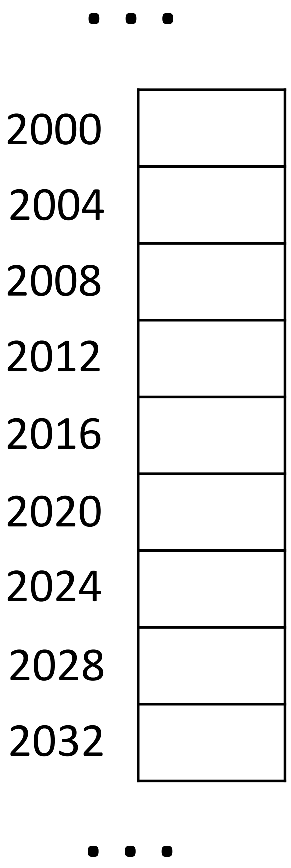
Tutorial TAs:
AmirHossein Sojoodi & Shayan Noei

Adapted from slides by
Bryony Schonewille & Shayan Noei

Today’s Session

- Brief Review of
  - Pointers
  - Structures
  - Unions
  - Enums

What are pointers?



. . .

2000

2004

2008

2012

2016

2020

2024

2028

2032

. . .

Memory is organized as a set of locations with consecutive addresses. Smallest location is a byte with its own address. Bytes are small, so items like integers use multiple bytes – in our case 4 – for storage.

Suppose we have the following declarations:

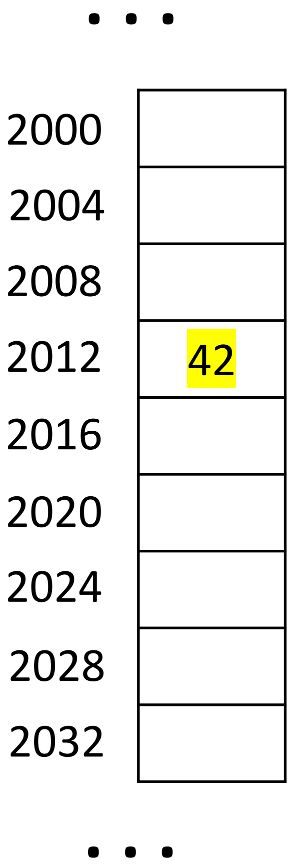
```
int  x;
int  *p;
```

Suppose also that the compiler arranged to use location 2012 for x and location 2016 for p.

What happens for

```
x = 42;
p = &x;
*p = 50;
```

What are pointers?



. . .	
2000	
2004	
2008	
2012	42
2016	
2020	
2024	
2028	
2032	
. . .	

Memory is organized as a set of locations with consecutive addresses. Smallest location is a byte with its own address. Bytes are small, so items like integers use multiple bytes – in our case 4 – for storage.

Suppose we have the following declarations:

```
int  x;
int  *p;
```

Suppose also that the compiler arranged to use location 2012 for x and location 2016 for p.

What happens for

```
x = 42;
p = &x;
*p = 50;
```

What are pointers?

. . .

2000	
2004	
2008	
2012	42
2016	2012
2020	
2024	
2028	
2032	

. . .

Memory is organized as a set of locations with consecutive addresses. Smallest location is a byte with its own address. Bytes are small, so items like integers use multiple bytes – in our case 4 – for storage.

Suppose we have the following declarations:

```
int  x;
int  *p;
```

Suppose also that the compiler arranged to use location 2012 for x and location 2016 for p.

What happens for

```
x = 42;
p = &x;
*p = 50;
```

What are pointers?

. . .

2000	
2004	
2008	
2012	42
2016	2012
2020	
2024	
2028	
2032	

. . .

Memory is organized as a set of locations with consecutive addresses. Smallest location is a byte with its own address. Bytes are small, so items like integers use multiple bytes – in our case 4 – for storage.

Suppose we have the following declarations:

```
int  x;
int  *p;
```

Suppose also that the compiler arranged to use location 2012 for x and location 2016 for p.

What happens for

```
x = 42;
p = &x;
*p = 50;
```

Check contents of location 2016 – where p is. Value there (2012) is the address used to store the value of the expression.

What are pointers?

. . .

2000	
2004	
2008	
2012	50
2016	2012
2020	
2024	
2028	
2032	

. . .

Memory is organized as a set of locations with consecutive addresses. Smallest location is a byte with its own address. Bytes are small, so items like integers use multiple bytes – in our case 4 – for storage.

Suppose we have the following declarations:

```
int  x;
int  *p;
```

Suppose also that the compiler arranged to use location 2012 for x and location 2016 for p.

What happens for

```
x = 42;
p = &x;
*p = 50;
```

Check contents of location 2016 – where p is. Value there (2012) is the address used to store the value of the expression.

What are pointers? Summary

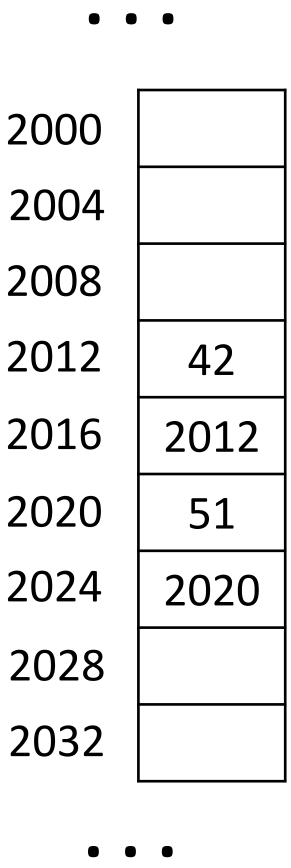
- A pointer is a variable that holds the address of some other piece of data
- Pointer can be assigned a value using the “address-of” operator: **p = &x;**
- Pointer can be used to access the data it points to, using the dereference operator: ***p = *p + 1;**

Pointers and Arrays

- C language shortcut – name of an array on its own is equivalent (syntax-wise) as the address of its first element.

```
int     x[10];
int *p;
p  =  x;                    // same as p  =  &x[0];
```

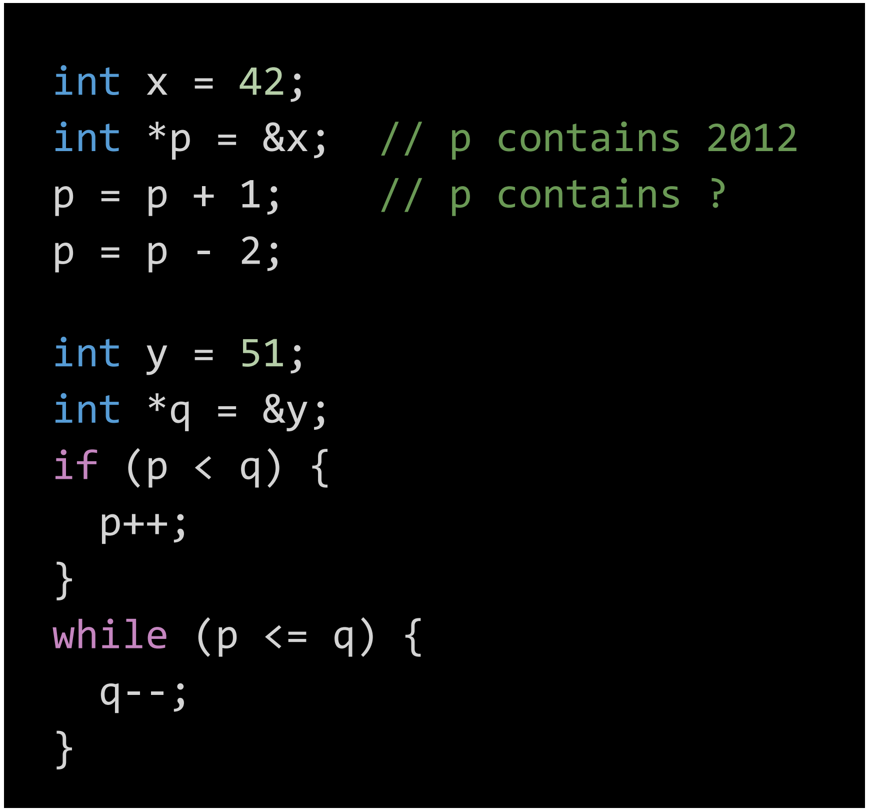
Pointer Arithmetic



. . .

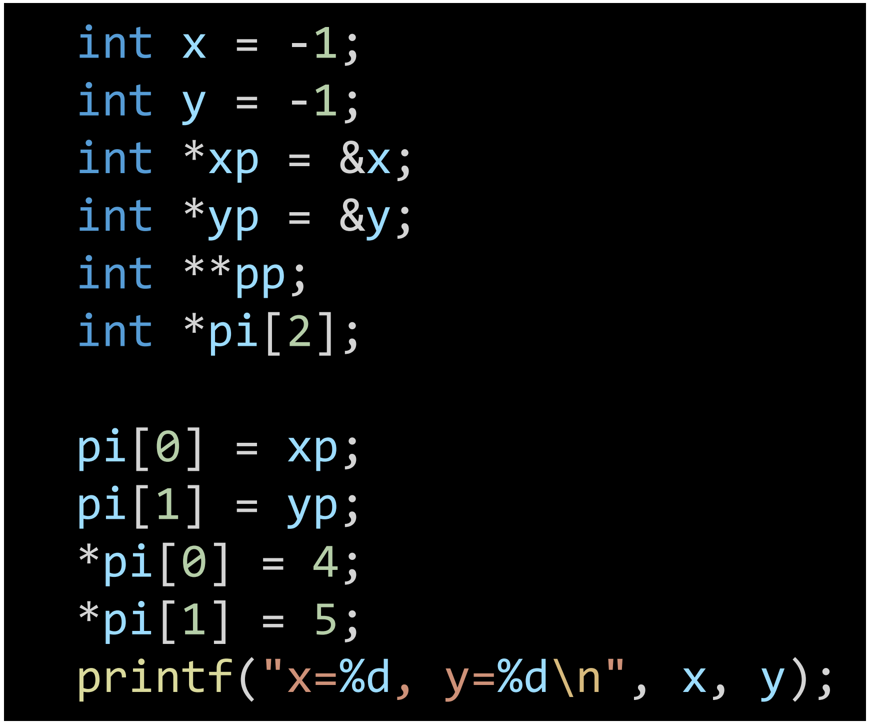
2000	
2004	
2008	
2012	42
2016	2012
2020	51
2024	2020
2028	
2032	

. . .

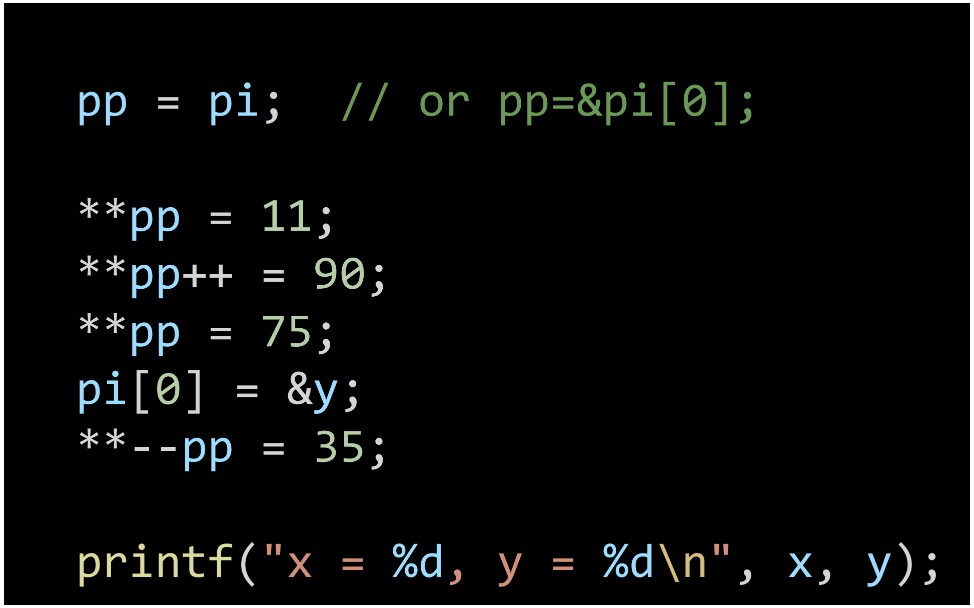


```
int x = 42;
int *p = &x;  // p contains 2012
p = p + 1;    // p contains ?
p = p - 2;

int y = 51;
int *q = &y;
if (p < q) {
  p++;
}
while (p <= q) {
  q--;
}
```

Fun with Pointers!



```
int x = -1;
int y = -1;
int *xp = &x;
int *yp = &y;
int **pp;
int *pi[2];

pi[0] = xp;
pi[1] = yp;
*pi[0] = 4;
*pi[1] = 5;
printf("x=%d, y=%d\n", x, y);
```



```
pp = pi;  // or pp=&pi[0];

**pp = 11;
**pp++ = 90;
**pp = 75;
pi[0] = &y;
**--pp = 35;

printf("x = %d, y = %d\n", x, y);
```

https://github.com/amirsojoodi/ELEC278

Pointers - types

- Every data type in C language has an associated pointer type.

```
int  *a;
float *b;
…
void *d;  What does this mean?!
```

- We also have pointer to functions! Let’s see an example

Pointers - advantages

- What is the point in using them?
  - Direct access to the memory
  - Data movement
    - Move the data itself around – Bad!
    - Move the pointer to the data around – Good!
  - Data allocation and deallocation – dynamically (let's see an example)
  - Return multiple values from a function without using return (more on this later)

Pointers - disadvantages

- Research shows that three of the top-ten sources of programming errors are pointer related.
  - Dereferencing the NULL or uninitialized pointer
  - Using a pointer to malloc()ed memory after it has been free()ed
  - Walking a pointer off the end of a piece of memory it was supposed to be pointing to

Better light a candle than curse the darkness.