Personal SE

Computer Memory
Addresses
C Pointers

Computer Memory Organization

- Memory is a bucket of *bytes*.

Computer Memory Organization

- Memory is a bucket of bytes.
  - Each byte is 8 bits wide.

Computer Memory Organization

- Memory is a bucket of bytes.
  - Each byte is 8 bits wide.
  - Question: How many distinct values can a byte of data hold?

Computer Memory Organization

- Memory is a bucket of bytes.
  - Each byte is 8 bits wide.
  - Question: How many distinct values can a byte of data hold?
  - Bytes can be combined into larger units:

Half-words (shorts)	16 bits	65,536 combinations	
Words (ints)	32 bits	≈ 4×10^9	≈ 4 billion
Double words (long)	64 bits	≈ 16×10^{18}	≈ 16 quadrillion

Computer Memory Organization

- Memory is a bucket of bytes.
  - Each byte is 8 bits wide.
  - Question: How many distinct values can a byte of data hold?
  - Bytes can be combined into larger units:

Half-words (shorts)	16 bits	65,536 combinations	
Words (ints)	32 bits	$\approx 4 \times 10^{9}$	$\approx$ 4 billion
Double words (long)	64 bits	$\approx 16 \times 10^{18}$	$\approx$ 16 quadrillion

- The bucket is actually an *array* of bytes:

Computer Memory Organization

- Memory is a bucket of bytes.
  - Each byte is 8 bits wide.
  - Question: How many distinct values can a byte of data hold?
  - Bytes can be combined into larger units:

Half-words (shorts)	16 bits	65,536 combinations	
Words (ints)	32 bits	$\approx 4 \times 10^{9}$	≈ 4 billion
Double words (long)	64 bits	$\approx 16 \times 10^{18}$	≈ 16 quadrillion

- The bucket is actually an array of bytes:
  - Think of it as an array named `memory`.

Computer Memory Organization

- Memory is a bucket of bytes.
  - Each byte is 8 bits wide.
  - Question: How many distinct values can a byte of data hold?
  - Bytes can be combined into larger units:

Half-words (shorts)	16 bits	65,536 combinations	
Words (ints)	32 bits	≈ 4×10^{9}	≈ 4 billion
Double words (long)	64 bits	≈ 16×10^{18}	≈ 16 quadrillion

- The bucket is actually an array of bytes:
  - Think of it as an array named `memory`.
  - Then `memory[ a ]` is the byte at index / location / address *a*.

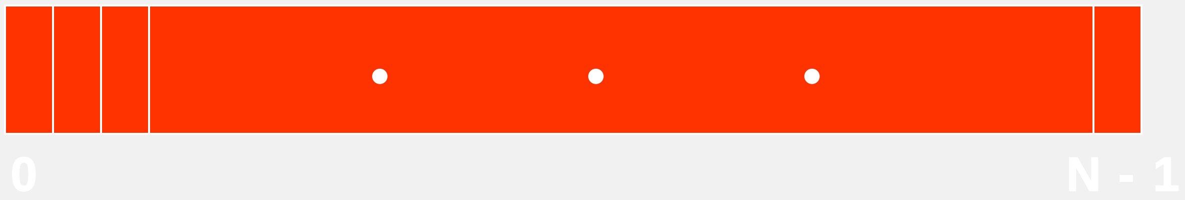
Computer Memory Organization

- Memory is a bucket of bytes.
  - Each byte is 8 bits wide.
  - Question: How many distinct values can a byte of data hold?
  - Bytes can be combined into larger units:

Half-words (shorts)	16 bits	65,536 combinations	
Words (ints)	32 bits	$\approx 4 \times 10^9$	≈ 4 billion
Double words (long)	64 bits	$\approx 16 \times 10^{18}$	≈ 16 quadrillion

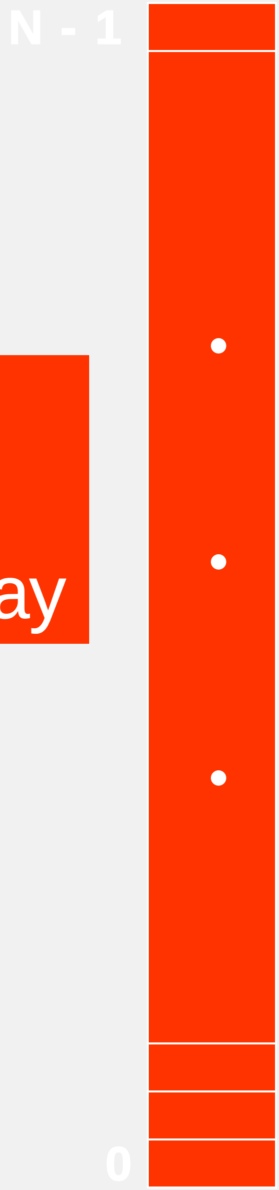
- The bucket is actually an array of bytes:
  - Think of it as an array named `memory`.
  - Then `memory[ a ]` is the byte at index / location / address a.
  - Normally the *addresses* run from 0 to some maximum.

Pictorially ... N byte Memory



Either way (horizontal or vertical) is fine.

The key is that memory is logically an array



What's In a Number?

- What does the hexadecimal number 0x4A6F65 mean?

What's In a Number?

- What does the hexadecimal number 0x4A6F65 mean?
- Possibilities:
  - It could be the decimal number 4,878,181
  - It could be the string "Joe"
    'J' = 0x4A, 'o' = 0x6F, 'e' = 0x65
  - It could be the address of the 4,878,181st byte in memory
  - It could be an instruction to, say, increment (op code = 0x4A) a location (address = 0x6F65) by 1

What's In a Number?

- What does the hexadecimal number 0x4A6F65 mean?
- Possibilities:
  - It could be the decimal number 4,878,181
  - It could be the string "Joe"
    'J' = 0x4A, 'o' = 0x6F, 'e' = 0x65
  - It could be the address of the 4,878,181st byte in memory
  - It could be an instruction to, say, increment (op code = 0x4A)
    a location (address = 0x6F65) by 1
- How do we know??????

What's In a Number?

- What does the hexadecimal number 0x4A6F65 mean?
- Possibilities:
  - It could be the decimal number 4,878,181
  - It could be the string "Joe"
    'J' = 0x4A, 'o' = 0x6F, 'e' = 0x65
  - It could be the address of the 4,878,181st byte in memory
  - It could be an instruction to, say, increment (op code = 0x4A) a location (address = 0x6F65) by 1
- How do we know??????
- We don't until we use it!

What's In a Number?

- What does the hexadecimal number 0x4A6F65 mean?
- Possibilities:
  - It could be the decimal number 4,878,181
  - It could be the string "Joe"
    'J' = 0x4A, 'o' = 0x6F, 'e' = 0x65
  - It could be the address of the 4,878,181st byte in memory
  - It could be an instruction to, say, increment (op code = 0x4A) a location (address = 0x6F65) by 1
- How do we know??????
- We don't until we use it!
  - If we send it to a printer, it's a string.
  - If we use it to access memory, it's an address.
  - If we fetch it as an instruction, it's an instruction.

Computer Numbers as Shape-Shifters

- The ability of numbers to "morph" their meaning is very powerful.
  - We can manipulate characters like numbers.
  - We can change instructions on the fly.
  - We can perform computation on addresses.

Danger Will Robinson! Danger!

- The ability of numbers to "morph" their meaning is very powerful.
  - We can manipulate characters like numbers.
  - We can change instructions on the fly.
  - We can perform computation on addresses.
- BUT: What if we use a number other than intended:
  - We get run-time errors (using an integer as an address).
  - We get hard-to-fix bugs (executing data as instructions).
  - We get weird printout (sending addresses to a printer).

Spiderman Is A "C" Programmer

- The ability of numbers to "morph" their meaning is very powerful.
  - We can manipulate characters like numbers.
  - We can change instructions on the fly.
  - We can perform computation on addresses.
- BUT: What if we use a number other than intended:
  - We get run-time errors (using an integer as an address).
  - We get hard-to-fix bugs (executing data as instructions).
  - We get weird printout (sending addresses to a printer).

With great power

comes great responsibility.

Pointers in C

- Consider the following two declarations:

```
int i ;
int *ip ;
```

Pointers in C

- Consider the following two declarations:

```
int i ;
int *ip ;
```

"*" says that ip is a pointer, not an integer

Pointers in C

- Consider the following two declarations:

```
int i ;
int *ip ;
```

The "*" is attached to the variable, not the type

Pointers in C

- Consider the following two declarations:

```
int i ;
int *ip ;
```

int i, *ip ;

Equivalent to these two declarations

Pointers in C

- Consider the following two declarations:

```
int i ;
int *ip ;
```

- On most systems, both allocate 32 bits for `i` and `ip`.

Pointers in C

- Consider the following two declarations:

```
int i ;
int *ip ;
```

- On most systems, both allocate 32 bits for `i` and `ip`.
- The difference?
  - `i`'s contents are treated as an *integer* – just a number.
  - `ip`'s contents are treated as an *address* (where an integer can be found).

Pointers in C

- Consider the following two declarations:

```
int i ;
int *ip ;
```

- On most systems, both allocate 32 bits for `i` and `ip`.
- The difference?
  - `i`'s contents are treated as an integer.
    - All we can manipulate is the integer value in i.
  - `ip`'s contents are treated as an address (where an integer can be found).
    - We can manipulate the address (make it point elsewhere).
    - We can manipulate the integer at the current address.

A Short Example

```
double x = 3.14159 ;
double y = 2.71828 ;
double *dp ;
```

NAME	ADDR	VALUE
x	108	3.14159
y	116	2.71828
dp	124	???????

A Short Example

```
double x = 3.14159 ;
double y = 2.71828 ;
double *dp ;

dp = &x ;
```

NAME	ADDR	VALUE
x	108	3.14159
y	116	2.71828
dp	124	???????

A Short Example

```
double x = 3.14159 ;
double y = 2.71828 ;
double *dp ;

dp = &x ;
```

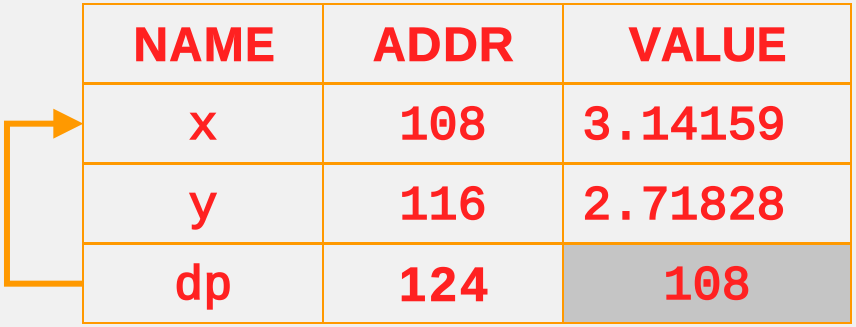
NAME	ADDR	VALUE
x	108	3.14159
y	116	2.71828
dp	124	???????

& = "address of"
The address of a variable is a pointer to the variable's type

A Short Example – The Effect

```
double x = 3.14159 ;
double y = 2.71828 ;
double *dp ;

dp = &x ;
```

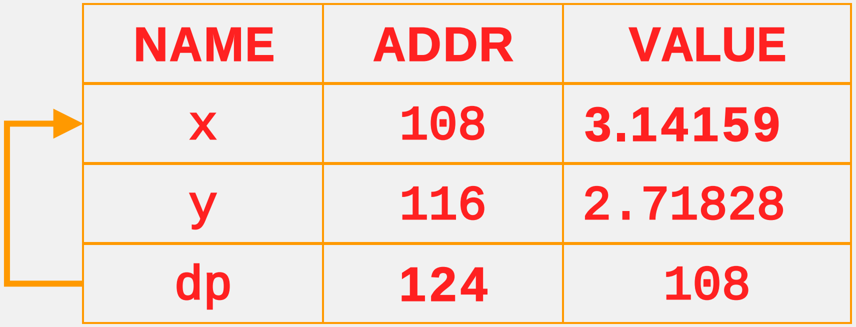


NAME	ADDR	VALUE
x	108	3.14159
y	116	2.71828
dp	124	108

A Short Example

```
double x = 3.14159 ;
double y = 2.71828 ;
double *dp ;

dp = &x ;
x = *dp * 2.0 ;
```

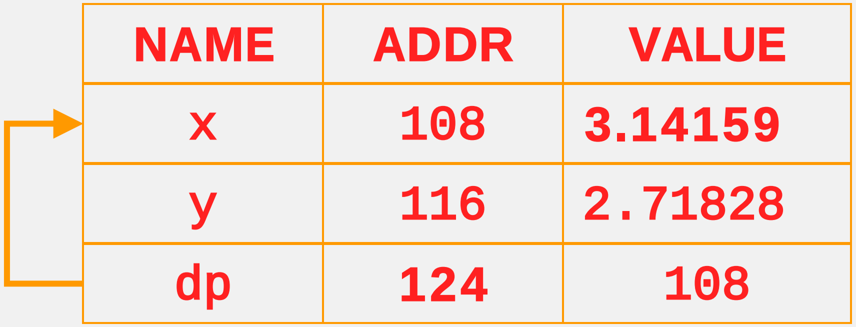


NAME	ADDR	VALUE
x	108	3.14159
y	116	2.71828
dp	124	108

A Short Example

```
double x = 3.14159 ;
double y = 2.71828 ;
double *dp ;

dp = &x ;
x = *dp * 2.0 ;
```



NAME	ADDR	VALUE
x	108	3.14159
y	116	2.71828
dp	124	108

* = "dereference"
The value the pointer addresses,
not the pointer itself

A Short Example – The Effect

```
double x = 3.14159 ;
double y = 2.71828 ;
double *dp ;

dp = &x ;
x = *dp * 2.0 ; // same as x = x * 2.0
```

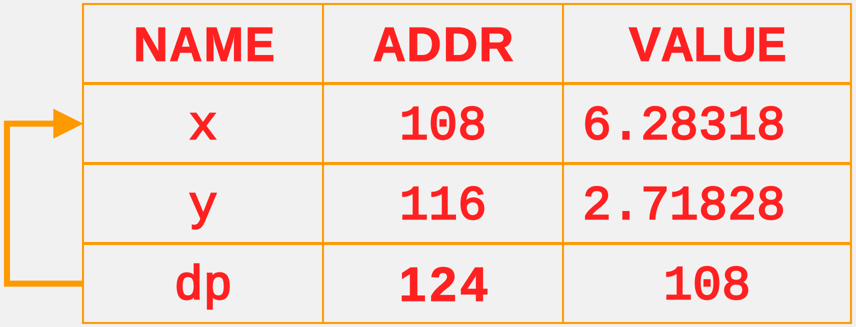
NAME	ADDR	VALUE
x	108	6.28318
y	116	2.71828
dp	124	108

A Short Example

```
double x = 3.14159 ;
double y = 2.71828 ;
double *dp ;

dp = &x ;
x = *dp * 2.0 ; // same as x = x * 2.0

dp = &y ;
```



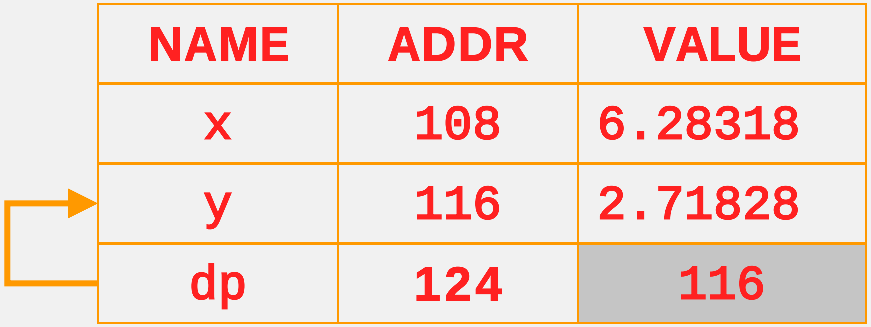
NAME	ADDR	VALUE
x	108	6.28318
y	116	2.71828
dp	124	108

A Short Example – The Effect

```
double x = 3.14159 ;
double y = 2.71828 ;
double *dp ;

dp = &x ;
x = *dp * 2.0 ; // same as x = x * 2.0

dp = &y ;
```



NAME	ADDR	VALUE
x	108	6.28318
y	116	2.71828
dp	124	116

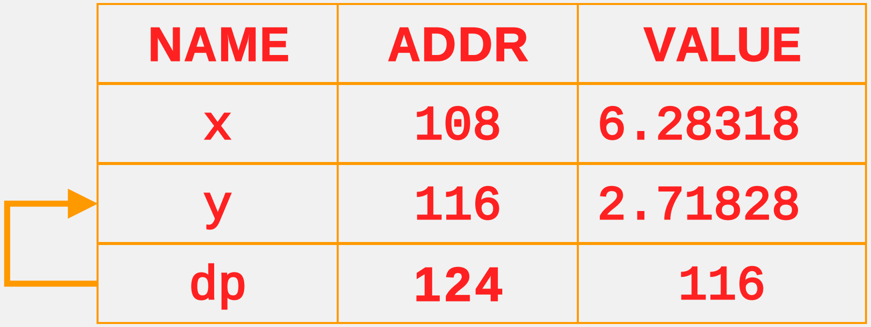
A Short Example

```
double x = 3.14159 ;
double y = 2.71828 ;
double *dp ;

dp = &x ;
x = *dp * 2.0 ; // same as x = x * 2.0

dp = &y ;

*dp += x ;
```



NAME	ADDR	VALUE
x	108	6.28318
y	116	2.71828
dp	124	116

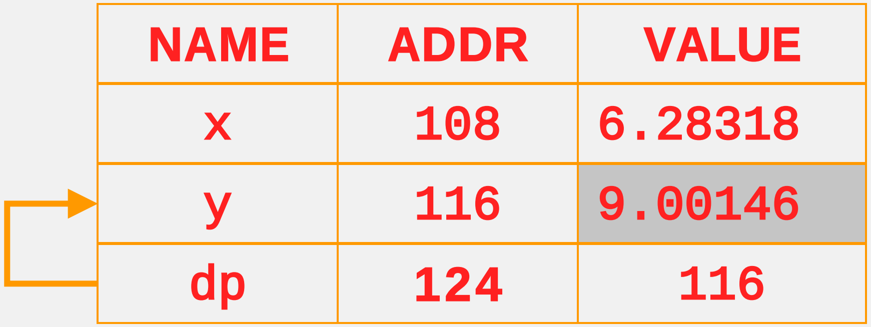
A Short Example – The Effect

```
double x = 3.14159 ;
double y = 2.71828 ;
double *dp ;

dp = &x ;
x = *dp * 2.0 ; // same as x = x * 2.0

dp = &y ;

*dp += x ;
```



NAME	ADDR	VALUE
x	108	6.28318
y	116	9.00146
dp	124	116

Pointers – Reference Parameters

Pointers – Reference Parameters

```
// Swap – the wrong way

void swap( grade_entry x, grade_entry y ) {
    grade_entry temp ;

    temp = x ;    x = y ;    y = temp ;

    return ;
}
```

Pointers – Reference Parameters

```
// Swap – the wrong way
void swap( grade_entry x, grade_entry y ) {
    grade_entry temp ;

    temp = x ;    x = y ;    y = temp ;

    return ;
}
// Swap – the right way

void swap( grade_entry *x, grade_entry *y ) {
    grade_entry temp ;

    temp = *x ;  *x = *y ;  *y = temp ;

    return ;
}
```

Pointers – Call by Reference

Pointers – Call by Reference

```
// Array element exchange the wrong way

swap( grade_list[ i ], grade_list[ j ] ) ;
```

Pointers – Call by Reference

```
// Array element exchange the wrong way
swap( grade_list[ i ], grade_list[ j ] ) ;
```

```
// Array element exchange the right way
swap( &grade_list[ i ], &grade_list[ j ] ) ;
```