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?
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 \( \approx 4 \times 10^9 \approx 4 \text{ billion} \)
  - Double words (long) 64 bits \( \approx 16 \times 10^{18} \approx 16 \text{ quadrillion} \)
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 $\approx 4 \times 10^9 \approx 4$ billion
    - Double words (long) 64 bits $\approx 16 \times 10^{18} \approx 16$ 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 \( \approx 4 \times 10^9 \approx 4 \text{ billion} \)
    - Double words (long) 64 bits \( \approx 16 \times 10^{18} \approx 16 \text{ quadrillion} \)

- **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`. 
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 \( \approx 4 \times 10^9 \approx 4 \text{ billion} \)
  - Double words (long) 64 bits \( \approx 16 \times 10^{18} \approx 16 \text{ quadrillion} \)

The bucket is actually an array of bytes:
- Think of it as an array named \text{memory}.
- Then \text{memory}[a] is the byte at index / location / address a.
- Normally the \textit{addresses} run from 0 to some maximum.
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 \texttt{0xA6F65} mean?
- Possibilities:
  - It could be the decimal number \texttt{4,878,181}
  - It could be the string "Joe"
    \texttt{'J' = 0xA, 'o' = 0xF, 'e' = 0x65}
  - It could be the address of the \texttt{4,878,181}\textsuperscript{st} byte in memory
  - It could be an instruction to, say, increment (op code = \texttt{0xA}) a location (address = \texttt{0xF65}) by 1
What's In a Number?

- What does the hexadecimal number \textcolor{red}{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 4,878,181\textsuperscript{st} 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.
Consider the following two declarations:

```c
int i;
int *ip;
```
Pointers in C

Consider the following two declarations:

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

"*" says that ip is a pointer, not an integer.
Pointers in C

Consider the following two declarations:

```c
int i;
int *p;
```

The "*" is attached to the variable, not the type.
Consider the following two declarations:

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

Equivalent to these two declarations:

```c
int i, *ip;
```
Pointers in C

- Consider the following two declarations:
  ```c
  int i;
  int *ip;
  ```
- On most systems, both allocate 32 bits for `i` and `ip`. 
Pointers in C

- Consider the following two declarations:
  ```c
  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).
Consider the following two declarations:

```c
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;

<table>
<thead>
<tr>
<th>NAME</th>
<th>ADDR</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>108</td>
<td>3.14159</td>
</tr>
<tr>
<td>y</td>
<td>116</td>
<td>2.71828</td>
</tr>
<tr>
<td>dp</td>
<td>124</td>
<td>???????</td>
</tr>
</tbody>
</table>

A Short Example

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

<table>
<thead>
<tr>
<th>NAME</th>
<th>ADDR</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>108</td>
<td>3.14159</td>
</tr>
<tr>
<td>y</td>
<td>116</td>
<td>2.71828</td>
</tr>
<tr>
<td>dp</td>
<td>124</td>
<td>???????</td>
</tr>
</tbody>
</table>
A Short Example

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

dp = &x;

<table>
<thead>
<tr>
<th>NAME</th>
<th>ADDR</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>108</td>
<td>3.14159</td>
</tr>
<tr>
<td>y</td>
<td>116</td>
<td>2.71828</td>
</tr>
<tr>
<td>dp</td>
<td>124</td>
<td>???????</td>
</tr>
</tbody>
</table>

& = "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;
A Short Example

double x = 3.14159;
double y = 2.71828;
double *dp;
dp = &x;
x = *dp * 2.0;

<table>
<thead>
<tr>
<th>NAME</th>
<th>ADDR</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>108</td>
<td>3.14159</td>
</tr>
<tr>
<td>y</td>
<td>116</td>
<td>2.71828</td>
</tr>
<tr>
<td>dp</td>
<td>124</td>
<td>108</td>
</tr>
</tbody>
</table>
A Short Example

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

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

<table>
<thead>
<tr>
<th>NAME</th>
<th>ADDR</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>108</td>
<td>3.14159</td>
</tr>
<tr>
<td>y</td>
<td>116</td>
<td>2.71828</td>
</tr>
<tr>
<td>dp</td>
<td>124</td>
<td>108</td>
</tr>
</tbody>
</table>

* = "dereference"
The value the pointer addresses, not the pointer itself
double x = 3.14159;
double y = 2.71828;
double *dp;

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

<table>
<thead>
<tr>
<th>NAME</th>
<th>ADDR</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>108</td>
<td>6.28318</td>
</tr>
<tr>
<td>y</td>
<td>116</td>
<td>2.71828</td>
</tr>
<tr>
<td>dp</td>
<td>124</td>
<td>108</td>
</tr>
</tbody>
</table>
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;
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;

<table>
<thead>
<tr>
<th>NAME</th>
<th>ADDR</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>108</td>
<td>6.28318</td>
</tr>
<tr>
<td>y</td>
<td>116</td>
<td>2.71828</td>
</tr>
<tr>
<td>dp</td>
<td>124</td>
<td>116</td>
</tr>
</tbody>
</table>
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;

<table>
<thead>
<tr>
<th>NAME</th>
<th>ADDR</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>108</td>
<td>6.28318</td>
</tr>
<tr>
<td>y</td>
<td>116</td>
<td>9.00146</td>
</tr>
<tr>
<td>dp</td>
<td>124</td>
<td>116</td>
</tr>
</tbody>
</table>
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 ] );