Memory Layout

• Linear sequence of values
• Each value has an address
• Similar to an array
• When you create a variable $x$
  - it will be associated with a memory address
  - at that address it will store the value of the variable

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>34542</td>
<td>8</td>
</tr>
<tr>
<td>34543</td>
<td>23</td>
</tr>
<tr>
<td>34544</td>
<td>0</td>
</tr>
<tr>
<td>34545</td>
<td>32</td>
</tr>
<tr>
<td>34546</td>
<td>127</td>
</tr>
<tr>
<td>34547</td>
<td>3432</td>
</tr>
<tr>
<td>34548</td>
<td>80</td>
</tr>
<tr>
<td>34549</td>
<td>111</td>
</tr>
<tr>
<td>34550</td>
<td>0</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
What is a Pointer?

• A pointer is when a memory location's value contains a memory location.

• Sometimes it is easier to tell someone where it is instead of what is there.

• For example, if you wanted to pass an array of 10 billion numbers to a function how would you do it?

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>34542</td>
<td>8</td>
</tr>
<tr>
<td>34543</td>
<td>34548</td>
</tr>
<tr>
<td>34544</td>
<td>0</td>
</tr>
<tr>
<td>34545</td>
<td>32</td>
</tr>
<tr>
<td>34546</td>
<td>127</td>
</tr>
<tr>
<td>34547</td>
<td>3432</td>
</tr>
<tr>
<td>34548</td>
<td>80</td>
</tr>
<tr>
<td>34549</td>
<td>111</td>
</tr>
<tr>
<td>34550</td>
<td>0</td>
</tr>
<tr>
<td>...</td>
<td>---</td>
</tr>
</tbody>
</table>
When we write

```c
int values[5] = {1, 3, 5, 2, 3};
```

the program will place the 5 values into a sequence of memory locations.

But the variable `values` is actually a pointer to the memory location of the first element in that sequence.
Arrays and Pointers

Using the table on the right, values would equal 12516.

Therefore, writing

values[2]

is the same as writing

(values+2)[0]

both equal 5
To create a pointer which points to memory location of an integer:

\[
\text{int}\,*\, x;
\]

Writing

\[
x = 5;
\]

**does not** set the value of the integer \( x \) to 5, but rather it states that \( x \) is pointing to an integer at memory location 5.
How to Get a Memory Address

- When using pointers we often want to get another variable's memory address.
- This can be done using the & operator:

```c
int x = 5;
int* p = &x;
```

- After these two lines of code, the pointer `p` would contain the memory address of the variable `x`. 
Dereference a Pointer

• To dereference a pointer means to get the value of the memory location it is pointing to.

• To do this you use the * operator:

```cpp
int x = 5;
int* p = &x;

cout << x << endl;
cout << *p << endl;
```
Extra Credit

What number would the following code print out to the screen and why:

```cpp
int values[5] = {1, 3, 5, 2, 3};
cout << *(values+3) << endl;
```

(first to answer gets 5 points on homework 4)
Pointers as Function Arguments

One of the most typical uses of pointers are for using them as function arguments. What is the difference between these two programs:

```c
void f(int x)
{
    x = 5;
}

int main()
{
    int x = 0;
    f(x);
    cout << x << endl;
}
```

```c
void f(int* x)
{
    *x = 5;
}

int main()
{
    int x = 0;
    f(&x);
    cout << x << endl;
}
```
Arrays as Function Arguments

- To pass an array to a function, you must pass it as a pointer.
- Since you cannot determine the size of an array from the pointer, you should also pass in the size of the array.

```c
int sum_array( int* array, int size )
{
    int sum = 0;

    for( int i = 0; i < size; i++ )
        sum += array[i];

    return sum;
}
```
Arrays as Function Arguments

To call the function from the main it would look like this:

```c
int main()
{
    int array[7] = { 3, 6, -2, 124, 4, -32, 3 };

    int sum = sum_array( array, 7 );

    cout << "Sum = " << sum << endl;

    return 0;
}
```
Arrays as Return Values

• Pointers can also be used to return an array of values from a function.

• This often helps to overcome the limitation of only being able to return one value from a function.

• Take a look at the dBodyGetPosition function in ODE.
Function Pointers

• It is even possible to have a pointer to a function.
• This is often used in handling errors, GUI environments, handling signals, etc.
• We will see this being used for drawing and collision detection in ODE.
• Using function pointers gives us the ability to call a function from only knowing its memory location.
void error(string message)
{
    cout << "Error: " << message << endl;
    exit(0);
}

int main()
{
    complex( error );

    return 0;
}
Function Pointers Example

```c
void complex( void (*errorFunction)(string) )
{
    ...

    if( div == 0 )
        errorFunction(“Trying to divide by zero.”);

    ...
}
```