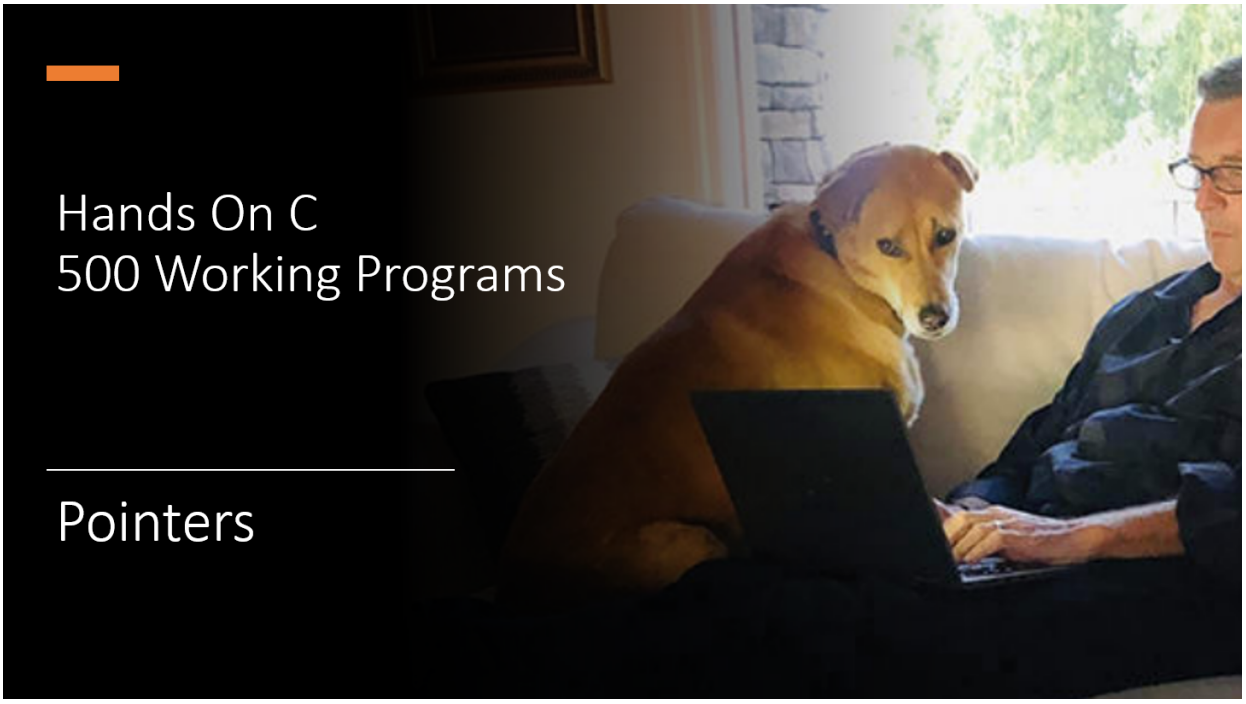

A Pointer is an Address

In [1]:
```c
#include <stdio.h>

int main(void)
{
   int value = 1000;

   printf("The address of value in RAM is %lu\n",(long unsigned int) &value);
}
```

```
The address of value in RAM is 140725710632564
```

In [2]:
```c
#include <stdio.h>

int main(void)
{
   int value = 1000;
   int b = 2;

   printf("The address of value in RAM is %lu\n",(long unsigned int) &value);
   printf("sizeof value is %ld bytes\n", sizeof(value));
   printf("Next variable should start at %ld\n", (long unsigned int) &value + siz
   printf("The address of b in RAM is %lu\n",(long unsigned int) &b);
}
```

```
The address of value in RAM is 140735132690000
sizeof value is 4 bytes
Next variable should start at 140735132690004
The address of b in RAM is 140735132690004
```

In [3]:
```c
#include <stdio.h>

int main(void)
{
   int value = 1000;
   int b = 2;

   printf("The address of value in RAM is %lu\n",(long unsigned int) &value);
   printf("The value of value is %d\n", value);
   printf("The address of b in RAM is %lu\n",(long unsigned int) &b);
   printf("The value of b is %d\n", b);
}
```

```
The address of value in RAM is 140725116965024
The value of value is 1000
The address of b in RAM is 140725116965028
The value of b is 2
```

Declaring a Pointer Variable

In [4]:
```c
#include <stdio.h>

int main(void)
{
  int value = 1000;

  int *pointer;

  pointer = &value;

  printf("Value %d\n", *pointer);
}
```

```
Value 1000
```

In [5]:
```c
#include <stdio.h>

int main(void)
{
  int value = 1000;

  int *pointer;

  pointer = &value;  // assign the address
  *pointer = 5000;   // assign a value to the address

  printf("Value pointed to by *pointer is %d\n", *pointer);
  printf("value %d\n", value);
}
```

```
Value pointed to by *pointer is 5000
value 5000
```

In [6]:
```c
#include <stdio.h>

int main(void)
{
  float pi = 3.14;

  float *pointer = &pi;

  printf("pi %f\n", pi);

  *pointer = 22.0/7.0;
  printf("pi %f\n", pi);
}
```

```
pi 3.140000
pi 3.142857
```

C Treats an Array as a Pointer

In [7]:
```c
#include <stdio.h>

int main(void)
{
   int array[] = { 1, 2, 3, 4, 5};

   printf("The address of array is %ld\n", (long unsigned int) array);
}
```

```
The address of array is 140726186322704
```

In [8]:
```c
#include <stdio.h>

int main(void)
{
   int array[5] = { 1, 2, 3, 4, 5};
   char string[] = "Hello, world";

   printf("The address of array is %ld\n", (long unsigned int) array);
   printf("The address of string is %ld\n", (long unsigned int) string);
}
```

```
The address of array is 140731464169632
The address of string is 140731464169659
```

In [9]:
```c
#include <stdio.h>

void showString(char *string)
{
   while (*string)
     printf("%c", *string++);
}

int main(void)
{
   char string[] = "Hello, world\n";

   showString(string);
}
```

```
Hello, world
```

In [10]:
```c
#include <stdio.h>

void showString(char *string)
{
   while (*string)
     printf("%c", *string++);
}

int main(void)
{
   char *string = "Hello, world\n";

   showString(string);
}
```

```
Hello, world
```

Using a Pointer to an Array of float

In [11]:
```
#include <stdio.h>

void showArray(float *values, int size)
{
   for (int i = 0; i < size; ++i)
     printf("%f ", *values++);
}

int main(void)
{
  float values[5] = { 1.1, 2.2, 3.3, 4.4, 5.5 };
  showArray(values, 5);
}
```

```
1.100000 2.200000 3.300000 4.400000 5.500000
```

Understanding an Array of Pointers

In [12]:
```
#include <stdio.h>

int main(void)
{
   char *strings[5] = { "Hello, ", "World", "Pointers", "are", "easy!" };

   for (int i = 0; i < 5; ++i)
     printf("%s\n", strings[i]);
}
```

```
Hello,
World
Pointers
are
easy!
```

In [13]:
```
#include <stdio.h>

int main(int argc, char *argv[])
{
   for (int i = 0; i < argc; ++i)
     printf("%s\n", argv[i]);
}
```

```
/tmp/tmpgbz_9rte.out
```

In [ ]:
```

```

Creating a Pointer to Pointer

In [14]:
```c
#include <stdio.h>

int main(void)
{
   char *strings[6] = { "Hello, ", "World", "Pointers", "are", "easy!" };
   char **pointer = strings;

   while (*pointer)
     printf("%s\n", *pointer++);
}
```

```
Hello,
World
Pointers
are
easy!
```

In [15]:
```c
#include <stdio.h>

int main(int argc, char **argv)
{
   while (*argv)
     printf("%s ", *argv++);
}
```

```
/tmp/tmpf4zk7hm8.out
```

Getting Crazy a Pointer to a Pointer to a Pointer

In [16]:
```c
#include <stdio.h>

int what_is_the_value(int ***ptr)
 {
   return(***ptr);
 }

int main(void)
 {
   int *level_1, **level_2, ***level_3, value = 1001;

   level_1 = &value;
   level_2 = &level_1;
   level_3 = &level_2;

   printf("The value is %d\n", what_is_the_value(level_3));
 }
```

```
The value is 1001
```

Why Pointers are Dangerous

In [17]:
```
#include <stdio.h>

int main(void)
{
  char string1[25] = "Hello, world";
  char string2[25] = "Kris Jamsa";

  char *pointer1 = string1;
  char *pointer2 = string2;

  while (*pointer1)  // find end of string1
    pointer1++;

  while (*pointer1++ = *pointer2++)  // append string2
    ;

  printf("%s\n", string1);
}
```

```
Hello, worldKris Jamsa
```

In [18]:
```
#include <stdio.h>

int main(void)
{
  char string1[] = "Hello, world";
  char string2[25] = "Kris Jamsa";

  char *pointer1 = string1;
  char *pointer2 = string2;

  while (*pointer1)  // find end of string1
    pointer1++;

  while (*pointer1++ = *pointer2++)  // append string2
    ;                                // error exceeds size of string1

  printf("%s\n", string1);
}
```

```
Hello, worldKris Jamsa
```

Using a Pointer to a Function

In [19]:
```
#include <stdio.h>

void one(void)
{
   printf("1111");
}

void two(void)
{
   printf("2222");
}

void callAFunction(void (*function)(void))
{
  function();
}

int main(void)
{
  callAFunction(two);
  callAFunction(one);
}
```

```
22221111
```

In [20]:
```c
#include <stdio.h>

int add(int a, int b)
{
   return(a+b);
}

int mult(int a, int b)
{
   return(a*b);
}

void callAFunction(int (*function)(int, int), int a, int b)
{
  printf("%d ", function(a, b));
}

int main(void)
{
  callAFunction(add, 1, 3);
  callAFunction(mult, 2, 4);
}
```

```
4 8
```

What's Coming

```
Pointers to Strings
Pointers to Structures
```

```
Pointers to Functions
Dynamic Memory Allocation
```

What You will Learn Next

```
C programs make extensive use of pointers, particulary with character strings.
In the next lesson, you will learn how to build your own string library of
commonly-used string operations.

int strlen(char *string)
{
  int count = 0;

  while (*string++)
    count++;

  return(count);
 }
```