Example 1-using the address operator &

```
computer$ gcc practice.c
computer$ ./a.out
apples=3
address of apples variable: 0x7fff51b2fa7c
```

```
#include <stdio.h>

int main(int argc, char**argv)
{
        int apples=3;
        printf("apples=%d\n",apples);
        printf("address of apples variable: %p\n",&apples); /*using the address operator to get the address of the apples variable.  Notice the %p format specifier used to print out an address (like we use %c to print a char)*/
}
```

Example 2-using a pointer

```
computer$ gcc practice.c
computer$ ./a.out
apples addy=0x7fff5f276a7c
```

```
#include <stdio.h>

int main(int argc, char**argv)
{
        int apples=3;
        int *a_ptr=&apples; /*using the address operator to get the address of the apples variable.  This operation return the address and stores it in the pointer variable a_ptr*/

        printf("apples addy=%p\n",a_ptr); /*we can now print out the value in this pointer (an address).  Remember the address is stored as a value in the variable just like any other variable.  For example int c=4; 4 is just a value in the int variable.  A pointer is the same concept it just holds an address.*/

}
```

Note: do not do something like this:

*int *ptr;*
**ptr=3; /*you are dereferencing a pointer, but there is nothing in that pointer (no address). So when you dereference, there is no address stored in ptr and you will get an error (called a segmentation fault) */*

You can do this:

int num=3;
*int *ptr=#*
**ptr=7; /*this works because there is an address in ptr to dereference (while you did not have one in the previous example). You are changing the value of num to 7*/*

Example 3-using the dereference operator *

```
computer$ gcc practice.c
computer$ ./a.out
apples addy=0x7fff57253a7c
the value at apples is: 3
```

```
#include <stdio.h>

int main(int argc, char**argv)
{
        int apples=3;
        int *a_ptr=&apples;

        printf("apples addy=%p\n",a_ptr);
        printf("the value at apples is: %d\n", *a_ptr); /*notice I am using the dereference operator here. This operator takes the value at the pointer (an address), goes to that address and returns the value at that address*/

}
```

Example 4-showing pointers pt II

```
computer$ gcc practice.c
Computers-MacBook-Air:C computer$ ./a.out
letter: f
letter address (two ways): 0x7fff56f91a7f,0x7fff56f91a7f

price: 4.500000
price address (two ways): 0x7fff56f91a6c, 0x7fff56f91a6c
```

```
#include <stdio.h>

int main(int argc, char**argv)
{
        char letter='f';
        char *c_ptr=&letter;

        float price=4.50;
        float *p_ptr=&price;

        printf("letter: %c\n", *c_ptr);
        printf("letter address (two ways): %p,%p\n\n", c_ptr, &letter);

        printf("price: %f\n", *p_ptr);
        printf("price address (two ways): %p, %p\n", p_ptr, &price);
```

}

Example 5-arrays and pointers

```
computer$ gcc practice.c
Computers-MacBook-Air:C computer$ ./a.out
words array: 0x7fff543f9a7d
0x7fff543f9a7d, 0x7fff543f9a7e, 0x7fff543f9a7f
c, a, t
```

```
#include <stdio.h>

int main(int argc, char**argv)
{
                char words[]={'c','a','t'};

                printf("words array: %p\n", words); /*notice this address is the same address as the letter c (first letter) below*/

/*the address operator & is used to get the address of each element of the array:*/
                char *ptr1=&words[0];
                char *ptr2=&words[1];
                char *ptr3=&words[2];

/*we can now print the address (first line) and value using the deref operator on the second line. Note that we can also use &words[0] directly instead of a pointer.  Notice the address  of the letter c is the same as the address of the array itself*/

                printf("%p, %p, %p\n", ptr1, ptr2, ptr3);
                printf("%c, %c, %c\n", *ptr1, *ptr2, *ptr3);
}
```

Example 6-scanf

```
computer$gcc practice.c
computer$ ./a.out
Original value in num: 4
88
Value in num now: 88
99
Value in num now: 99
```

```
#include <stdio.h>

int main(int argc, char**argv)
{
        int num=4;
        int *n_ptr=&num;
```

```
        printf("Original value in num: %d\n", num);

        /*notice you can use a pointer in scanf as the second argument*/
        scanf("%d", n_ptr);
        printf("Value in num now: %d\n", num);

        /*you can also use the address operator*/
        scanf("%d", &num);
        printf("Value in num now: %d\n", num);

}
```

Example 7-pointers as parameters to functions

```
computer$ gcc practice.c
computer$ ./a.out
Address of letter is: 0x7fff598e3a7f
Letter actually is: f

Address of letter is: 0x7fff598e3a7f
Letter actually is: f
```

```
#include <stdio.h>

void foo(char *lptr)
{
        printf("Address of letter is: %p\n", lptr);
        printf("Letter actually is: %c\n\n", *lptr);
}


int main(int argc, char**argv)
{
        char lett1='f';
        char *lett_ptr=&lett1;

        foo(&lett1);
        foo(lett_ptr);
}
```

Example 8- sizeof operator

```
computer$ gcc sizeof.c
computer$ ./a.out


*****Info:
size of char: 1 byte              size of ptr_char: 8 bytes
size of double: 8 bytes           size of ptr_double: 8 bytes
```

```
size of float: 4 bytes          size of ptr_float: 8 bytes
size of int: 4 bytes            size of ptr_int: 8 bytes
```

Notes: %lu format specifier means unsigned long (a data type-you can look it up)
From C99 and up we can use the %zu format specifier

```
#include <stdio.h>

int main(int argc, char **argv)
{

        printf("\n\n*****Info: \n");

        char letter='a';
        char *ptr_char=&letter;


        printf("size of char: %lu byte          size of ptr_char: %lu bytes\n", sizeof(char), sizeof(char *));


        double stuff=4;
        double *ptr_double=&stuff;

        printf("size of double: %lu bytes       size of ptr_double: %lu bytes\n", sizeof(double),
sizeof(double *));


        float more_stuff=4;
        float *ptr_float=&more_stuff;

        printf("size of float: %lu bytes        size of ptr_float: %lu bytes\n", sizeof(float), sizeof(float *));


        int num=3;
        int *ptr_int=&num;

        printf("size of int: %lu bytes          size of ptr_int: %lu bytes \n", sizeof(int), sizeof(int *)); /*on
my machine the size of my pointer is 8 bytes-so I could hold 2^64 different addresses in a pointer*/

}
```

```c
#include "stdio.h"

void example_one(int value[])
{
  printf("Size in function: %lu\n", sizeof(value));
}

void example_two(int* val)
{
  printf("Size in function two: %lu\n", sizeof(val));
}

int main(int argc, char ** argv)
{

   int values[3];
   printf("In main: %lu\n", sizeof(values));

   example_one(values);
   example_two(values);

}
```

Output:

```
computer$ ./a.out
In main: 12
Size in function: 8
Size in function two: 8
```

Notice when passing an array to a function, when using the sizeof operator, we get the size of a pointer…the array is passed as an address, NOT the actual array itself

Using sizeof on the array itself, we get the actual size (12 bytes-3 ints)

You even get a warning about this:

```
(base) Computers-MacBook-Air:C computer$ gcc practice.c
practice.c:10:44: warning: sizeof on array function parameter
 will return size of
      'int *' instead of 'int []' [-Wsizeof-array-argument]
   printf("Size in function: %lu\n", sizeof(value));
                                            ^
practice.c:8:22: note: declared here
void example_one(int value[])
                     ^
1 warning generated.
```

Example 9- Sample Problem

Clarice is an amateur bird watcher. After weeks of watching, she has noticed that she can figure out the number of birds she will see based on the number of cats she sees in the neighborhood. She noticed that she sees three times as many birds as cats. Also, she noticed that on Saturday she sees an extra 5 birds (in addition to the total mentioned above).

```
computer$ gcc -o bird birds.c
computer$ ./bird
Enter number of cats seen today: 5
Enter the day: Saturday
Saturday
Total number of birds: 20
```

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

/*remember an array is called by an address, so we can pass the array in directly for the first argument*/
int total_birds(char *day, int *c)
{
        printf("%s",day);
```

```
	int bird=*c *3; /* we are dereferencing c by using the dereferencing operator: *c and then multiplying by 3*/

	/*check if the day entered was saturday or not-strcmp returns 0 only if the two arguments match*/
	if(strcmp(day, "Saturday\n")==0||strcmp(day, "saturday\n")==0)
	{
		bird=bird+5;
	}
	return bird; /*returns the number of birds seen*/
}


int main(int argc, char **argv)
{
	/*we will get info a little differently-we will use fgets for all input.  for the int, we will convert from a string to int using the atoi function (include stdlib.h header for atoi function)*/
	char cats[3];
	char day[10];
	int i;
	printf("Enter number of cats seen today: ");
	fgets(cats, 3, stdin); /*getting input using fgets.  1st arg: the char array to put our input, 2nd arg: size of char array, 3rd arg input from the keyboard*/
	int num=atoi(cats); /*since we got our answer as a string, we need to convert it to an integer using the atoi function*/

	printf("Enter the day: ");
	fgets(day, 10, stdin); /*same as above*/


	int birds=total_birds(day, &num); /*we could have created an int pointer to pass the info in but we are using the address operator*/
	printf("Total number of birds: %d\n", birds);

}
```
