Day 18 Getting More from Functions

passing by value- function is passed a copy of the argument's value

```
3 steps:
1. argument expression is evaluated
2. result is copied onto the stack, a temporary storage space in memory
3. the function retrieves the argument's value from the stack
```

if a variable is passed as the argument, code in the function cannot modify the value of the variable

IF nearly every function will need to modify the data directly, can use global variable

otherwise pay attention:

variable contents are copied onto stack during execution so function cannot modify original value

passing by reference- pass a pointer to the argument variable rather than the value of the variable itself

limited space on the stack- should not pass large items like structures

advantage and disadvantage to pass by reference- may unintentionally change some data

COULD just modify by using the function's return value, but that limits you to modifying only a single value

ensure prototype and definition understand you will be passing a pointer

for calling:

```
prototype
void by_ref(int *a, int *b, int *c);
definition
by_ref(&x, &y, &z);
```

could write a function that retrieves some values by reference and others by value

Type void Pointers

generic pointer that can point to any type of data object

most common use is in declaring function parameters

could pass type int one time ,type float another etc.

can use a typecast to tell function what data type- cannot dereference a pointer until function knows the exact data type

```
void *val;
(type *) pval
tell it its an int:
(int *) pval
to derefrence:
*(int *) pval
```

could do something like this:

```
void half(void *pval, char type);
```

then integrate into function

Using Functions that have a Variable number of arguments

can write own functions perform like scanf and printf

must include stdarg.h

first declare fixed parameters. then use ellipsis at end of parameter list to indicate that zero or more additional arguments are passed to the function

you tell it how many arguments will be passed

one of the functions fixed arguments will be used for the additional arguments

to create a function that accepts different variable types, must devise method of passing information- like character code w/ switch statement

what you need to use for this:

```
va_list a pointer to data type
va_start() a macro used to initialize the argument list
va_arg() a macro used to retrieve each argument, in turn, from the variable list
va_end() macro used to clean up when all arguments have been retrieved
```

must follow these steps:

```
1. declare a pointer variable of type va_list. usually called arg_ptr

2. call va_start() passing it arg_ptr and **name of last fixed argument**

3. to retrieve each argument call va_arg() passing pointer and data type of next argument. r

4. when done call va_end()
```

Example:

```
#include <stdio.h>
#include <stdargt.h>

float average(int num, ...);

int main( void )
{
    float x;

    x = average(*numbers*)
    printf(first average is x)

    x = average(*second set*)
    printf(second average is x)
}

float average(int num, ...)
{
    va_list arg_ptr;
    int count, total = 0;

    va_start(arg_ptr, num);

    for(count = 0; count <num; count++)
        total += va_arg( arg_ptr, int);

    va_end(arg_ptr);

    return ((float)total/num);
}
```

returns averages of each list- see pg 525

Functions that return a pointer

```
type *func(parameter_list);
```