Day 15 Pointers: Beyond the Basics

pointer to pointer- variable whose value is the address of a pointer

```
int myVar = 12;
int *ptr = &myVar;
int **ptr2ptr = &ptr;
```

use double indirection operator when accessing pointed to varaible:

```
**ptr2ptr = 12;
```

assigns value of 12 to myVar

most common use is for arrays of pointers

Pointers and Multi-Dimensional Arrays

review:

```
int multi[2][4]
```

```
1. multi contains 2 elements
2. each of these 2 elements contains 4 elements
3. each of the 4 elements is a type int
```

ARRAY OF ARRAYS

name of multidimensional array is a pointer to first array element

multi[0] would be pointer to multi[0][0] or first element of multidimensional array

the array name followed by n pairs of brackets evaluates as array data (data stored in the specified array element)

array name followed by fewer than n pairs of brackets evaluates as pointer to an array element

```
sizeof(multi) = 32
sizeof(multi[0] = 16
sizeof(multi[0][0]) = 4
```

to access data at multi[0][0]:

```
multi[0][0]
*multi[0]
**multi
```

array with n dimensions has elements that are arrays of n-1 dimensions

to declare a pointer to an element of multi (can point to 4 element integer array):

```
int (*ptr)[4]
```

to point at first element of multi:

```
prt = multi;
```

use typecast (ex:(int *)) when handling pouinters to pointers of arrays

Working with Arrays of Pointers

most common use of array of pointers with strings

start of string indicated by poitner to first character

by declaring and initializing an array of pointers to type char can access and manipulate large number of strings using pointer array

review: string must have space allocated for it. whether at compilation at declaration or at runtime with malloc()

declare an array of 10 pointers to type char:

```
char *message[10];
```

```
char *message[10] = { "one" , "two" , "three" }
```

allocates 10 element array named message- each pointer to type char allocates space in memory for the three strings. each with terminating null character initializes array indexes point at first character of each string

much easier to pass array of pointers to function then pass several strings. when pass that array to a function passing a pointer (array name) to a pointer (first array element)

review: malloc() returns a pointer

```
int (*b)[12]; //pointer to array of 12 integers
int *c[12];  // array of 12 pointers to integers
```

Working with Pointers to Functions

when program runs, code for each function loaded into memory starting at specific address- pointer to function holds starting address- entry point

more flexible way of calling a function

```
type (*ptr_to_func)(parameter_list)
```

must be declared but also initialized to point

```
float square(float x);  //prototpye
float (*ptr)(float x);  //pointer declaration
float square(float x);  //function definition
ptr = square        //initialize ptr to square
answer = ptr(x)     //call function using pointer
```

function name without () is pointer to function

but that is a constant and cannot be changed- why the pointers are useful

declare a pointer to a function that takes no arguments and returns a character:

```
char (\*func)();
```

declare a function returns a pointer to a character

```
char \*func()
```

Linked Lists pg416

single linked lists

contained in a structure- add pointers to structure links to other instances to generate linking

last element in list identified by pointer assigned value of NULL

```
instance1 -> instance2 -> instance3 -> NULL
```

create HEAD pointer as intermediate step

add an element to beginning of list

```
1. create instance of structure, allocating memory space malloc()
2. set next pointer of new element to the current value of the head pointer
3. make the head pointer point to the new element
```

adding an element to the end of the list

```
1. create instance of structure, allocating memory space malloc()
2. set the next pointer in the last element to point to the
 new element (whose address is returned by malloc())
3. set the next pointer in the new element to NULL to signal that i
it is the last item in the list
```

adding an element to the middle of the list

```
1. in the list, locate the exisitng element that the new elementi
 will be placed after- marker element
2. create an instance of your structure, allocating memory malloc()
3. set the next pointer of the marker element to point to the new
 element whose address is returned by malloc()
```

4. set the next pointer of the new element that the marker element used to point to

deleting element from a list

1. to delete the first element set the head pointer to point toi
 the second element in the list
2. to delete the last element set the next pointer of the next-to-lasti
 element to NULL
3. to delete any other element set the next pointer of the element before i
the one being delete to point to the element after the one being deleted