Notes 11/17

if you declare  
*arr[10] = {1, 2, 3, 4, 5, 6, 7 ,8, 9, 10}  
int\* ptr = arr;*

*ptr* will return the memory address of the first element of *arr.*cout << \*ptr; //this returns the value pointed to by ptr. \* is the reference operator.

Pointers hold the location in memory of a declared value.

arr[0] == \*(arr+0)  
  
You can usethe *sizeof()* operator on any datatype to return the size of the memory address for the input.

Thus, if you want to use pointers to traverse and step through arrays, you can say

for(int I = 0; I < 10; i++)  
{  
 cout << \*(ptr + i)  
}  
**or**  
for(int I = 0; I < 10; i++)  
{  
 cout << \*(arr + i)  
}

example from project 6

int main()  
{  
 int arr[3] = {5, 10, 15};  
 int \*ptr = arr;  
 \*ptr = 10;  
 \*ptr + 1 = 20;  
 ptr += 2;  
 ptr[0] = 30;  
 while(ptr >= arr)  
 {  
 ptr--;  
 cout << “ “ << \*ptr;  
 }  
}

The problem is supposed to output  
  
10  
20  
30  
  
But it doesn’t. We have to fix it.  
  
int main()  
{  
 int arr[3] = {5, 10, 15};  
 int \*ptr = arr;  
 \*ptr = 10; //this sets the first value of the array = to 10. The array is now {10, 10, 15}  
 \*ptr + 1 = 20; //this is a compiler error! Need to set parentheses around (ptr + 1). If so, {10, 20,15}  
 ptr += 2; //increases the memory address being pointed to by 2. The pointer now points to the 15.  
 ptr[0] = 30; //same thing as saying \*(ptr + 0). So the array is now {10, 20, 30}.  
 while(ptr >= arr) //while the memory address of ptr is greater than the mem address of arr[0]  
 {  
 ptr--; //step back 1 element towards arr[0]  
 cout << “ “ << \*ptr; //this outputs the value pointed to   
 } //since ptr- - is before the cout, the last step of the loop will give you random values.  
}

This would output  
\_30\_20\_10\_*garbage*  
  
What if you changed the while loop to this?  
  
while (arr <= ptr)  
{  
 cout << \*arr << endl;  
 arr++;  
}

This would work because ptr is equal to the last value of arr at the end of the function. It would output  
10  
20  
30  
  
You could also use a for loop:  
  
for(int I = 0; I < 3; i++)  
{  
cout << \*(arr + i) << endl;  
}  
  
This would return the exact same output.  
  
Arrays can be used just as pointers are once they have been declared. By default they point to the first value in the array; if you increment them it will step through the array just as if you stepped through a[k] with k++.  
  
Structs can be used to get around the “one function, one return value” limitation.  
By the way, DON’T FORGET THE SEMICOLON.  
  
Structs can reference themselves inside their parameters. Like so:  
  
Struct Person  
{  
 string name;  
 int age;  
 string ss;  
 Person friends[150];  
 int numFriends;  
}**; 🡨🡨🡨**If you wanted to print out all of a person’s friends,   
  
void print(Person p)  
{  
 cout << p.name;  
 for (int I = 0; I < p.numFriends; i++)  
 {  
 cout << p.friends[i].name; //referencing a struct within a struct  
 }  
}  
A function to create Person values:  
  
Person create(string name, int age, string ss)  
{  
 Person p;  
 p.name = name;  
 p.age = age;  
 p.ss = ss;  
 return p;  
}  
  
**friends[0].name == (\*friends).name == friends ->name** //how to reference the first value of name in an array of friends.  
  
You can step through like this:  
Person \* ptr = friends;  
ptr++;  
ptr -> name;  
  
You can use the sizeof(datatype) operator on a struct. This is its best functionality because the memory used by struct can be confusing and different from struct to struct. Also, recall that   
Person \* ptr = friends; //ptr points to a Person variable  
ptr++; //this will increment ptr as such: ptr = ptr + sizeof(Person)