Notes on Pointers and Structs: (Sorry, I realise Game State Manager very hard to explain through notes)

Pointers

Pointers are variables whose main purpose is to point to another variable and allow access to that variable's value. The variable that the pointer is pointing to is called "Pointee".

Syntax:
Datatype of the Pointee * *ID* = &x; * Must be initialized with the same type

What does the pointer store?
The pointer stores the address of the variable that it is pointing to. The pointer must always be initialized with the address of the variable that it is pointing to.

Example:
Int x = 5;
int * ptr = &x; <- Initialize the pointer with the address
ptr; <- This is the address of the pointee
ptr = 0x100; <- This is changing who the pointer is pointing to. It is now pointing to the address of 0x100

How do I get access to the value of the pointee?
The "*" is known as the dereference operator. When used with the pointer, it allows access to the variable.

Example:
Int x = 5;
Int * ptr = &x;
*ptr; <- This gets the value of the pointee (which is 5)
*ptr = 4; <- Changes the value of the pointee. So now x has a value of 4

More examples:
Int x = 5;
Int y = 2;
Int * ptr1 = &x;
Int * ptr2 = &y;
ptr1 = ptr2; <- This change the pointer 'ptr1' to point at y as the address it stores is now address of y

So, what is the use of pointers?
Pointers are usually used in functions to allow values that are passed in to have a connection with the original variables. By using pointers, you can pass variables into function and whatever is done inside, will affect the original variables.

Example of Pointers Usage:

```
void Foo (int a)          <- Essentially, int a = main::x; This only assigns the value of x into a. If a is
{                         changed, it does not affect the value of main::x
  a = 10;
}

void Goo (int * a)        <- Essentially, int * a = &main::x; It ensures that a is a pointer to x and no
{                         copying is done
  *a = 8;
}

Int main()
{
  Int x = 5;
  Foo(x);                 <- The value of x after coming out of the function is still 5
  Goo(&x);                <- The value of x after coming out of the function is now 8

  return 0;
}
```

Function Pointers:

Syntax:
Return Type of Function (* *ID*) (*Parameters of Functions*) = &Function;

How do I read this?
Essentially, imagine a man starting at *ID*. It will always first walk to the right.
When it encounters a ')' while walking right, it is treated as a wall. He then walks left.
It then reads the "*", which means pointer.
When it encounters a '(' while walking left, it is treated as a wall. He then walks right.
When it encounters a '(' while walking right, it means it is a function.

Example:
Int (*ID)(int, float);
ID is a pointer that points to a function that takes in an integer and float, and returns an integer.

How do I call the function that the function pointer is pointing to:
Simply just call ID(*parameters*);

Example of Usage of Function Pointers: (This can be used in Game State Manager)

```
void foo(int x, double y)
{
  printf("Test1");
}

void goo(int x, double y)
{
  printf("Loser");
}

Int main()
{
  Void (*fp)(int, double) = &foo; <- Declaring a function pointer 'fp' and initializing with foo's address
  fp(5, 5.5);                     <- Calls the function foo. Prints out "Test1"
  fp = &goo;                      <- Change the pointee to goo
  fp(5, 5.5);                     <- Calls the function goo. Prints out "Loser"

  return 0;
}
```

Structs:

What are Structs?
Structs are user defined types (UDT) that can hold multiple members of different types.

How to declare a Struct type?

```
Struct Name
{
  Type Member1;
  Type Member2;
  …
};
```

Why do we use structs?
Instead of declaring multiple variables, structs allows us to declare only 1 variable that contains all the different members.

Example of a struct:

```
Struct Character
{
  char * name;
  int hp;
  int PosX;
  int PosY;
};
```

How do I declare a Struct variable?
First, ensure that you have declared the struct type first.
Syntax:
Struct Character John; <- *Struct Character* is a user-defined type

How do I access the members of the struct?
This is done using the "." member-of operator.
John.hp = 5; <- This is accessing the hp member of John and assigning 5 into the int hp

Example without Structs:

```
void UpdateCharacter(char** name, int*  hp, int* PosX, int* PosY);   <- Have to pass all the variables

int main()
{
  char* JohnName;        <- Without Structs, you have to declare each variable 1 by 1
  int JohnHp;               If you suddenly have to add a new variable, all the functions have to
  int JohnX;                change
  int JohnY;

  UpdateCharacter(&JohnName, &JohnHp, &JohnX, &JohnY);
}
```

Example with Structs

```
void UpdateCharacter(struct Character* char);   <- Only have to pass in the struct, instead of all the
                                                   variables
int main()
{
  struct Character John;  <-- Only need to declare the struct. It contains all the members.
                              If more members have to be added, the struct can be updated and
                              function does not need to change
  UpdateCharacter(&John);
}
```