Used in logic to Differentiate nodes						On Fresh save in Tough		Possible in Level				
Node Type	Level	Data Type	Code Notes	Size	Final Offset	Nodes from start	Nodes from end	Spawn amount?	Destroy Amount?		Node Type	"+0" "+d8" "+ec"
Display info	0x01	Gem Counter	level ends at 0	16bit	0x8d4	0		0			Display info	0x111328
Discrete Map	0x01	Map Pointer	Discrete check points, laps -> {0x0, 0x73, 0xe6, 0x153}	16bit	0x208	1		0			Discrete Map	0x129b38
Display info	0x02	Health	1 is full, 0 is empty	Float	0x7b4	1		~100			Object	0x192098
Hidden Counter	0x02	Scarab Counter	total destroyed in all rounds, 0 to 300	16bit	0x234		100				???	0x19ded0
Hidden Counter	0x02	Scarab Counter	total destroyed in rounds 1+2, resets for round 3 {0 > 200 > 0 > 100}	16bit	0x234		101				???	0x1a1118
Hidden Counter	0x02	Scarab Counter	total destroyed in round 1, resets for rounds 2+3 {0 > 100 > 0 > 200}	16bit	0x234		102				Hidden Counter	0x1b2868
Display info	0x03	Gem Counter	level ends at 0	16bit	0x8d4	0		~300?			Monkey	0x1ccdb8
Display info	0x05	Player Map Pointer	starts at 0 ish, ends at 1	Float	0xcc8	0		0			Phase	0x1e87e8
???	0x05	Enemy Map Pointer x2	starts at 0 ish, ends at 1	Float	???	???		???			???	0x1edd40
Display info	0x07	Health	1 is full, 0 is empty	Float	0x7b4	0		<60			???	0x1f7378
Object	0x07	Door	>0 when closed, <0 when open	Float	0xc		0		0		Gem	0x209a78
Papaya	0x09	Papaya in hand	0 if not held, 1 if held	16bit	0x2f8	0		~150			Coin	0x209a78
Papaya	0x09	Papaya not under tree	0 if moved from its spawn, 1 if it's where it spawns	16bit	0x74	0		~150			Scarab	0x20f770
Display info	0x09	Papaya Counter	level ends at 0	16bit	0x8d4	~10		~150			???	0x21f2b0
Display info	0x0a	Banana Counter		16bit	0x8d4	21		~30			Hidden Timer	0x24c990
Display info	0x0a	Monkey Counter	level ends at 0	16bit	0x8d6	21		~30			Target / Ring	0x250200
Monkey	0x0a	Monkey in Hand	0 if not held, 1 if held	16bit	0xa54	0-20		~30			Papaya / Boulder	0x276cf8
Monkey	0x0a	Monkey not Caught	0 if not in cage, 1 if in cage	16bit	0xa58	0-20		~30			Projectile	0x348b98
Display info	0x0b	Health	1 is full, 0 is empty	Float	0x7b4	4		<100			???	0x34e888
Display info	0x0b	Snowcone Counter		16bit	0x8d4	4		<100				
Phase	0x0b	Boss Phase	0 before starting, odd # minions, even # vunerable, 7 boss fight over	16bit	0x3fc		20		?			
Phase	0x0b	Boss Minion Counter	starts at 3 every odd phase, moves to even phase when hitting 0	16bit	0x400		20		?			
Phase	0x0b	???	Only used to help differentiate the above node	???	???		21		?			
Display info	0x0c	Seconds Timer	level fails at 0, get more time every time you hit the help ball	Float	0xcfc	0		0				
Display info	0x11	Engine Fuel	1 is full, 0 is empty, level fail on empty	Float	0x7b4	0		0				
Display info	0x11	Cheese Counter	level ends at 0	16bit	0x8d4	0		0				
Display info	0x1b	Floor Count	0 in yard, floors 1-3, 4 in angelica's tower (angelica exclusive)	16bit	0x7e8	0-4		1 (Angelica)				
Display info	0x1e	Carrot Counter	value stays in place while gameplay goes from 3 > 2	16bit	0x8d4	0		<25				
Display info	0x20	Ring Counter	value stays in place while gameplay goes from 3 > 2	16bit	0x1f4	0		A lot?				
Hidden Timer	0x20	Timer	Counts up from 0 to 60, game ends at 60	Float	0x9b0	1		A lot?				
Display info	0x24	Bashes Counter	value stays in place while gameplay goes from 3 > 2	16bit	0x8d4	0		<50?				

Starting pointer is held in memory at 0x50b944

Going forward in the list will have an addaddress chain that goes: Starting pointer > +10 > (+14) # of node times > +0 > +Final Offset
Going backward in the list will have an addaddress chain that goes: Starting pointer > +44 > +30 > (+10) # of node times > +0 > +Final Offset

The above going backward chain only works in some instances, other times it seems to access a whole other linked list? Very confusing, might need further research if other data that cannot be found is eventually needed (Looking at you Gamecube port)

Nodes listed from the start can be moved a node further in the list every time a new object spawns, and a node from the end can be moved closer to the end every time a node is destroyed

For instance, the logic to test if the snowcone counter
in level 0x1b changes from 0x20 to 0x1f at the start
of the level before nothing else spawns would look like

Node Type	0x111328	Size	16-bit
Nodes from start	4	Final Offset	0x8d4

	Mem	8-bit	Level	=	Value	0x1b	
AddAddress	Mem	32-bit	0x50b944				
AddAddress	Mem	32-bit	0x10				
AddAddress	Mem	32-bit	0x14				
AddAddress	Mem	32-bit	0x14				
AddAddress	Mem	32-bit	0x14				
Remember	Mem	32-bit	0x14				
AddAddress	Recall						
AddAddress	Mem	32-bit	0x0				
AddAddress	Mem	32-bit	0xd8				
AndNext	Mem	32-bit	0xec	=	Value	0x111328	
AddAddress	Recall						
AddAddress	Mem	32-bit	0x0				
AndNext	Delta	16-bit	0x8d4	=	Value	0x20	
AddAddress	Recall						
AddAddress	Mem	32-bit	0x0				
	Mem	16-bit	0x8d4	=	Value	0x1f	

Level Check
Access the Pointer
Access the Node
Testing Node Type
Delta Check
Mem Check

More specifically, access the pointer to the node
AddAddressing the 0x0 after the remember
is so we can check multiple nodes in a row
See to the right for an example

For checking several nodes in a row for the snowcone data,
the logic may look like this, with the purple chunk from
the example on the left collapsed into a single line

		Mem	8-bit	Level	=	Value	0x1b
	AddAddress	Mem	32-bit	0x50b944			
	Remember	Mem	32-bit	0x10			
0th node	Purple chunk copied from left with addHits in the last line						
	AddAddress	Recall					
	Remember	Mem	32-bit	0x14			
1st node	Purple chunk copied from left with addHits in the last line						
	AddAddress	Recall					
	Remember	Mem	32-bit	0x14			
2nd node	Purple chunk copied from left with addHits in the last line						
	AddAddress	Recall					
	Remember	Mem	32-bit	0x14			
3rd node	Purple chunk copied from left with addHits in the last line						
	AddAddress	Recall					
	Remember	Mem	32-bit	0x14			
4th node	Purple chunk copied from left with addHits in the last line						
	AddAddress	Recall					
	Remember	Mem	32-bit	0x14			
5th node	Purple chunk copied from left with addHits in the last line						
		Value	0x0	=	Value	0x1	(1)

The addhits are neccessary to act as an orNext chain without needing to split
this logic into seperate alt groups, as the Remember / Recall is impoartant to keep
logic length down to fit in the 64k character limit

This logic in the code is done with the function

```
comparison(data.levelIDLoaded, '=', 0x1b)
data.chainLinkedListRange(0, 5, ARRAY, true)
"0=1.1."
```

The inputs of the middle line being

0 - Start at node 0
5 - End at node 5
ARRAY - an array with an element per check you want to do per node, in this case 3.
The three bit of logic to the left that are green, orange, and red
true - We want to go forward through the list, not backwards