```
; =============== S U B R O U T I N E =======================

; Attributes: bp-based frame

; Node *__fastcall Node::Node(Node *__hidden this, int)
                public _ZN4NodeC2Ei ; weak
_ZN4NodeC2Ei    proc near               ; CODE XREF: Doubly

var_C           = dword ptr -0Ch
var_8           = qword ptr -8

; __unwind {
                push    rbp
                mov     rbp, rsp
                mov     [rbp+var_8], rdi
                mov     [rbp+var_C], esi
                mov     rax, [rbp+var_8]
                mov     edx, [rbp+var_C]
                mov     [rax], edx
                mov     rax, [rbp+var_8]
                mov     qword ptr [rax+8], 0
                mov     rax, [rbp+var_8]
                mov     qword ptr [rax+10h], 0
                nop
                pop     rbp
                retn
; } // starts at 126C
_ZN4NodeC2Ei    endp
```

(1) (2) (3) (4) (5)

1. Function Preamble
2. Allocate space for the next and prev pointers as well as the given val value for data.
    a. Note: I compiled this using standard g++ options. Despite there being two pointers defined within the source code, the assembly uses the defined qword to create them and only differentiate them in location rather than content. This is likely a minor compiler optimization as O0 is the default as specified by the GCC docs.
    b. RDI and ESI were also interesting to me, as I didn't initially understand why they were used for the future allocation within 3 and 4.
    c. After doing research, I hypothesize that since we are writing to the pointer values when adding nodes, the next and prev pointer allocations use RDIto denote that these will be written to (Pointers are also 8 byte values which is why they are using RDI rather than EDI). ESI is used to read in the 4 byte/32 bit int value VAL that we pass into creating the node object.

3. We move the ptr and int values stored in memory during (2) into registers to be utilized in (4). It's important to note that these are stored below the base pointer, denoting locals rather than args.
4. Stores the int value into the memory address pointed to by rax and then allocates 0 (nullptr values) to the prev and next pointers.
5. Function epilogue