My implementation for the vertex coloring problem is a greedy algorithm which starts from a given node and looks in the neighborhood for the colors used and chooses the minimum color not used. colors are represented with integers starting from 0

DATA STRUCTURES
node:
contains the name of the current node
it's color
it's adjacency list
a pointer to the next node

adj_list
contains a pointer to the node in the adjacency list
contains a poitner to the rest of the adjacency list

I used a list for nodes because the exact number of nodes was not known and an adjacency matrix couldn't have been used without allocating a lot of space beforehand for the maximum numbers of node possible. This code can be used for any number of nodes given there is memory available for each single node.

I used a list for adjacent nodes as well for the same reasons as above

main
i take the filename on command line
i read the graph from the file
initialize the first node of the graph with a sentinel
and then start to color nodes one by one
i then print the nodes and their colors while keeping track of the maximum color used
then output that value plus 1 to count the color 0
and then free the allocated memory

myfopen (filename,mode)
sees if the file opened successfully for the given mode and returns the pointer if it does otherwise closes the program

mymalloc(size)
sees if the malloc was successful and returns the pointer otherwise closes the program

pointerto (name,graph)
initializes a sentinel node if graph == null
looks for node named name in graph and returns it if found
otherwise creates a new node with color initialized to -1,appeneds it after the sentinel and before the next node returns that node

insert_adj(node1,node1)
add's node1 to node2's adjacency list head and vice versa

readGraph (filename, graph)
reads the edges stored in file one by one

uses pointerto to find the pointer to the nodes if it already exists or to create a new one if it doesn't and then uses insert_adj to add node 1 to node 2's adjacency list and vice versa

color_func(node)
returns if node is NULL
returns if node has already been colored
otherwise starting from color=0 checks if the color has been used if not then colors the node with that color otherwise increases the color and looks in the whole neighbourhood again to see if it has been used and then colors the node
then from there it uses the adjacency list to color all of the nodes recursively

printG(graph)
starts from the given node prints it's color and moves to the next node while keeping track of the highest color used
returns the highest color

freeG(graph)
if graph is null then it returns
frees the name of the current node
frees the adjacency list
frees the rest of the graph recursively
frees the current node

free_adj(adjacency_list)
returns if adjacency list is empty it returns
frees the rest of the adjacency list recursively
frees the current node

the changes have been commented in the code itself. There are no major changes. just a few changes to make the the code syntactically correct and work properly which were a little difficult to correct while programming with hand.

the code compiles

the code has been test on:
e.txt (minimum colors required 2) exam example
tc1.txt (minimum colors required 49) “zeroin.i.1.col” reference 1
tc2.txt (minimum colors required 30) “zeroin.i.2.col” reference 1
tc3.txt (minimum colors required 30) “zeroin.i.3.col” reference 1

the code has been checked for memory leaks and there are none

reference 1: https://mat.gsia.cmu.edu/COLOR/instances.html