```c
// Shamim Bin Zahid
// Lab 05 | FGP
// Roll: 43

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define MAX_CHAR 1000

char text[100];
Node *root = NULL;
Node *lastNewNode = NULL;
Node *activeNode = NULL;
int count=0;
int activeEdge = -1;
int activeLength = 0;
int remainingSuffixCount = 0;
int leafEnd = -1;
int *rootEnd = NULL;
int *splitEnd = NULL;
int size = -1;

struct SuffixTreeNode {
  struct SuffixTreeNode *children[MAX_CHAR];
  struct SuffixTreeNode *suffixLink;
  int start;
  int *end;
  int suffixIndex;
};

typedef struct SuffixTreeNode Node;

Node *newNode(int start, int *end){
  count++;
  Node *node =(Node*) malloc(sizeof(Node));
  int i;
  for (i = 0; i < MAX_CHAR; i++)
    node->children[i] = NULL;
  node->suffixLink = root;
  node->start = start;
  node->end = end;
  node->suffixIndex = -1;
  return node;
}

int edgeLength(Node *n) {
  return *(n->end) - (n->start) + 1;
}

int walkDown(Node *currNode){
  if (activeLength >= edgeLength(currNode)){
    activeEdge = (int)text[activeEdge+edgeLength(currNode)]-(int)' ';
    activeLength -= edgeLength(currNode);
    activeNode = currNode;
    return 1;
  }
  return 0;
}

```

```
void extendSuffixTree(int pos){
  leafEnd = pos;
  remainingSuffixCount++;
  lastNewNode = NULL;
  while(remainingSuffixCount > 0) {
    if (activeLength == 0) {
      activeEdge = (int)text[pos]-(int)' ';
    }
    if (activeNode->children[activeEdge] == NULL){
      activeNode->children[activeEdge] = newNode(pos, &leafEnd);
      if (lastNewNode != NULL) {
        lastNewNode->suffixLink = activeNode;
        lastNewNode = NULL;
      }
    }
    else {
      Node *next = activeNode->children[activeEdge];
      if (walkDown(next)) {
        continue;
      }
      if (text[next->start + activeLength] == text[pos]) {
        if(lastNewNode != NULL && activeNode != root) {
          lastNewNode->suffixLink = activeNode;
          lastNewNode = NULL;
        }
        activeLength++;
        break;
      }
      splitEnd = (int*) malloc(sizeof(int));
      *splitEnd = next->start + activeLength - 1;

      Node *split = newNode(next->start, splitEnd);
      activeNode->children[activeEdge] = split;

      split->children[(int)text[pos]-(int)' '] =
                  newNode(pos, &leafEnd);
      next->start += activeLength;
      split->children[activeEdge] = next;

      if (lastNewNode != NULL) {
        lastNewNode->suffixLink = split;
      }
      lastNewNode = split;
    }
    remainingSuffixCount--;
    if (activeNode == root && activeLength > 0) {
      activeLength--;
      activeEdge = (int)text[pos - remainingSuffixCount + 1]-(int)' ';
    }
    else if (activeNode != root) {
      activeNode = activeNode->suffixLink;
    }
  }
}

void printSequence(int i, int j) {
  for (int k = i; k <= j; k++)
    printf("%c", text[k]);
}

```

```
120 void setSuffixIndexByDFS(Node *n, int labelHeight) {
121   if (n == NULL) return;
122   if (n->start != -1) {
123     printSequence(n->start, *(n->end));
124   }
125   int leaf = 1;
126   for (int i = 0; i < MAX_CHAR; i++) {
127     if (n->children[i] != NULL) {
128       if (leaf == 1 && n->start != -1) printf(" [%d]\n", n->suffixIndex);
129       leaf = 0;
130       setSuffixIndexByDFS(n->children[i], labelHeight + edgeLength(n->children[i]));
131     }
132   }
133   if (leaf == 1) {
134     n->suffixIndex = size - labelHeight;
135     printf(" [%d]\n", n->suffixIndex);
136   }
137 }
138 
139 void freeSuffixTreeByPostOrder(Node *n) {
140   if (n == NULL) return;
141   for (int i = 0; i < MAX_CHAR; i++)
142     if (n->children[i] != NULL)
143       freeSuffixTreeByPostOrder(n->children[i]);
144   if (n->suffixIndex == -1) free(n->end);
145   free(n);
146 }
147 
148 void generateSuffixTree() {
149   size = strlen(text);
150   rootEnd = (int*) malloc(sizeof(int));
151   *rootEnd = - 1;
152   root = newNode(-1, rootEnd);
153   activeNode = root;
154   for (int i=0; i<size; i++)
155     extendSuffixTree(i);
156   int labelHeight = 0;
157   setSuffixIndexByDFS(root, labelHeight);
158   freeSuffixTreeByPostOrder(root);
159 }
160 
161 int main(int argc, char *argv[]) {
162   strcpy(text, "ATCATGTCATG");
163     generateSuffixTree();
164   printf("Number of nodes in suffix tree are %d\n",count);
165   return 0;
166 }
167 
```