

[*] Untitled1

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #define MAX_KEYS 2
4  typedef struct Node {
5      int keys[MAX_KEYS + 1];
6      struct Node *children[MAX_KEYS + 2];
7      int num_keys;
8      int is_leaf;
9  } Node;
10 Node* create_node(int is_leaf);
11 void traverse_tree(Node *root);
12 Node* insert(Node *root, int key);
13 void split_child(Node *parent, int index, Node *child);
14 void insert_non_full(Node *node, int key);
15 int main() {
16     Node *root = create_node(1);
17     int keys[] = {10, 20, 5, 6, 15, 30, 25, 40};
18     int num_keys = sizeof(keys) / sizeof(keys[0]);
19     for (int i = 0; i < num_keys; i++) {
20         root = insert(root, keys[i]);
21     }
22     printf("Tree traversal:\n");
23     traverse_tree(root);
24     return 0;
25 }
26 Node* create_node(int is_leaf) {
27     Node *new_node = (Node*)malloc(sizeof(Node));
28     new_node->is_leaf = is_leaf;
29     new_node->num_keys = 0;
30     for (int i = 0; i <= MAX_KEYS; i++) {
31         new_node->children[i] = NULL;
32     }
33     return new_node;
34 }
```

[*] Untitled1

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #define MAX_KEYS 2
4  typedef struct Node {
5      int keys[MAX_KEYS + 1];
6      struct Node *children[MAX_KEYS + 2];
7      int num_keys;
8      int is_leaf;
9  } Node;
10 Node* create_node(int is_leaf);
11 void traverse_tree(Node *root);
12 Node* insert(Node *root, int key);
13 void split_child(Node *parent, int index, Node *child);
14 void insert_non_full(Node *node, int key);
15 int main() {
16     Node *root = create_node(1);
17     int keys[] = {10, 20, 5, 6, 15, 30, 25, 40};
18     int num_keys = sizeof(keys) / sizeof(keys[0]);
19     for (int i = 0; i < num_keys; i++) {
20         root = insert(root, keys[i]);
21     }
22     printf("Tree traversal:\n");
23     traverse_tree(root);
24     return 0;
25 }
26 Node* create_node(int is_leaf) {
27     Node *new_node = (Node*)malloc(sizeof(Node));
28     new_node->is_leaf = is_leaf;
29     new_node->num_keys = 0;
30     for (int i = 0; i <= MAX_KEYS; i++) {
31         new_node->children[i] = NULL;
32     }
33     return new_node;
34 }
35 void traverse_tree(Node *root) {
36     if (root != NULL) {
```

```

63 parent->children[index + 1] = new_child;
64 parent->keys[index] = child->keys[1];
65 parent->num_keys++;
66 new_child->num_keys = 1;
67 child->num_keys = 1;
68 for (int i = 0; i < 2; i++) {
69     new_child->keys[i] = child->keys[i + 2];
70     new_child->children[i] = child->children[i + 2];
71 }
72 new_child->children[2] = child->children[4];
73 }
74 void insert_non_full(Node *node, int key) {
75     int i = node->num_keys - 1;
76
77     if (node->is_leaf) {
78         while (i >= 0 && key < node->keys[i]) {
79             node->keys[i + 1] = node->keys[i];
80             i--;
81         }
82         node->keys[i + 1] = key;
83         node->num_keys++;
84     } else {
85         while (i >= 0 && key < node->keys[i]) {
86             i--;
87         }
88         i++;
89         if (node->children[i]->num_keys == MAX_KEYS) {
90             split_child(node, i, node->children[i]);
91             if (key > node->keys[i]) {
92                 i++;
93             }
94         }
95         insert_non_full(node->children[i], key);
96     }
97 }

```

[*] Untitled1

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #define MAX_KEYS 3
4  #define MIN_KEYS 1
5  typedef struct Node {
6      int keys[MAX_KEYS + 1];
7      struct Node *children[MAX_KEYS + 2];
8      int num_keys;
9      int is_leaf;
10 } Node;
11 Node* create_node(int is_leaf);
12 void traverse_tree(Node *root);
13 Node* insert(Node *root, int key);
14 void split_child(Node *parent, int index, Node *child);
15 void insert_non_full(Node *node, int key);
16 void free_tree(Node *root);
17
18 int main() {
19     Node *root = create_node(1);
20     int keys[] = {10, 20, 5, 6, 15, 30, 25, 40, 50, 45, 55};
21     int num_keys = sizeof(keys) / sizeof(keys[0]);
22
23     for (int i = 0; i < num_keys; i++) {
24         root = insert(root, keys[i]);
25     }
26     printf("Tree traversal:\n");
27     traverse_tree(root);
28     free_tree(root);
29
30     return 0;
31 }
32 Node* create_node(int is_leaf) {
33     Node *new_node = (Node*)malloc(sizeof(Node));
34     new_node->is_leaf = is_leaf;
35     new_node->num_keys = 0;
36     for (int i = 0; i <= MAX_KEYS; i++) {
```


[*] Untitled1

```
37     new_node->children[i] = NULL;
38 }
39 return new_node;
40 }
41
42 void traverse_tree(Node *root) {
43     if (root != NULL) {
44         int i;
45         for (i = 0; i < root->num_keys; i++) {
46             if (!root->is_leaf) {
47                 traverse_tree(root->children[i]);
48             }
49             printf("%d ", root->keys[i]);
50         }
51         if (!root->is_leaf) {
52             traverse_tree(root->children[i]);
53         }
54     }
55 }
56 Node* insert(Node *root, int key) {
57     if (root->num_keys == MAX_KEYS) {
58         Node *new_root = create_node(0);
59         new_root->children[0] = root;
60         split_child(new_root, 0, root);
61         insert_non_full(new_root, key);
62         return new_root;
63     } else {
64         insert_non_full(root, key);
65         return root;
66     }
67 }
68 void split_child(Node *parent, int index, Node *child) {
69     Node *new_child = create_node(child->is_leaf);
70     parent->children[index + 1] = new_child;
71     parent->keys[index] = child->keys[0];
72     parent->num_keys++;
```

[*] Untitled1

```
71 parent->keys[index] = child->keys[1];
72 parent->num_keys++;
73 new_child->num_keys = 1;
74 child->num_keys = 1;
75 for (int i = 0; i < 2; i++) {
76     new_child->keys[i] = child->keys[i + 2];
77     new_child->children[i] = child->children[i + 2];
78 }
79 new_child->children[2] = child->children[4];
80 }
81 void insert_non_full(Node *node, int key) {
82     int i = node->num_keys - 1;
83     if (node->is_leaf) {
84         while (i >= 0 && key < node->keys[i]) {
85             i--;
86             i++;
87             if (node->children[i]->num_keys == MAX_KEYS) {
88                 split_child(node, i, node->children[i]);
89                 if (key > node->keys[i]) {
90                     i++;
91                 }
92             }
93             insert_non_full(node->children[i], key);
94         }
95     }
96 void free_tree(Node *root) {
97     if (root != NULL) {
98         if (!root->is_leaf) {
99             for (int i = 0; i <= root->num_keys; i++) {
100                 free_tree(root->children[i]);
101             }
102         }
103         free(root);
104     }
105 }
106 }
```

[*] Untitled1

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #define MAX_KEYS 4
4  #define MIN_KEYS 2
5  #define MAX_CHILDREN (MAX_KEYS + 1)
6  #define MIN_CHILDREN (MIN_KEYS + 1)
7  typedef struct Node {
8      int keys[MAX_KEYS];
9      struct Node *children[MAX_CHILDREN];
10     int num_keys;
11     int is_leaf;
12 } Node;
13 Node* create_node(int is_leaf);
14 void traverse_tree(Node *root);
15 Node* insert(Node *root, int key);
16 void split_child(Node *parent, int index, Node *child);
17 void insert_non_full(Node *node, int key);
18 void free_tree(Node *root);
19 int main() {
20     Node *root = create_node(1);
21     int keys[] = {10, 20, 5, 6, 15, 30, 25, 40, 50, 45, 55, 60, 70, 80};
22     int num_keys = sizeof(keys) / sizeof(keys[0]);
23     for (int i = 0; i < num_keys; i++) {
24         root = insert(root, keys[i]);
25     }
26     printf("Tree traversal:\n");
27     traverse_tree(root);
28     free_tree(root);
29     return 0;
30 }
31 Node* create_node(int is_leaf) {
32     Node *new_node = (Node*)malloc(sizeof(Node));
33     new_node->is_leaf = is_leaf;
34     new_node->num_keys = 0;
35     for (int i = 0; i < MAX_CHILDREN; i++) {
36         new_node->children[i] = NULL;
37     }
38     return new_node;
39 }
```

[*] Untitled1

```
37 }
38 void traverse_tree(Node *root) {
39     if (root != NULL) {
40         int i;
41         for (i = 0; i < root->num_keys; i++) {
42             if (!root->is_leaf) {
43                 traverse_tree(root->children[i]);
44             }
45             printf("%d ", root->keys[i]);
46         }
47         if (!root->is_leaf) {
48             traverse_tree(root->children[i]);
49         }
50     }
51 }
52 Node* insert(Node *root, int key)
53 {
54     if (root->num_keys == MAX_KEYS) {
55         Node *new_root = create_node(0);
56         new_root->children[0] = root;
57         split_child(new_root, 0, root);
58         insert_non_full(new_root, key);
59         return new_root;
60     } else {
61         insert_non_full(root, key);
62         return root;
63     }
64 }
65 void split_child(Node *parent, int index, Node *child) {
66     Node *new_child = create_node(child->is_leaf);
67     parent->children[index + 1] = new_child;
68     parent->keys[index] = child->keys[2];
69     parent->num_keys++;
70     new_child->num_keys = 2;
71     child->num_keys = 2;
72     for (int i = 0; i < 2; i++) {
73         new_child->keys[i] = child->keys[i + 3];
74     }
75 }
```


[*] Untitled1

```
73     new_child->children[i] = child->children[i + 3];
74 }
75 new_child->children[2] = child->children[5];
76 }
77 void insert_non_full(Node *node, int key) {
78     int i = node->num_keys - 1;
79     if (node->is_leaf) {
80         while (i >= 0 && key < node->keys[i]) {
81             node->keys[i + 1] = node->keys[i];
82             i--;
83         }
84         node->keys[i + 1] = key;
85         node->num_keys++;
86     } else {
87         while (i >= 0 && key < node->keys[i]) {
88             i--;
89         }
90         i++;
91         if (node->children[i]->num_keys == MAX_KEYS) {
92             split_child(node, i, node->children[i]);
93             if (key > node->keys[i]) {
94                 i++;
95             }
96         }
97         insert_non_full(node->children[i], key);
98     }
99 }
100 void free_tree(Node *root) {
101     if (root != NULL) {
102         if (!root->is_leaf) {
103             for (int i = 0; i <= root->num_keys; i++) {
104                 free_tree(root->children[i]);
105             }
106         }
107         free(root);
108     }
```