

```
1 // Implements a list of numbers with an array of fixed size
2
3 #include <stdio.h>
4
5 int main(void)
6 {
7     // List of size 3
8     int list[3];
9
10    // Initialize list with numbers
11    list[0] = 1;
12    list[1] = 2;
13    list[2] = 3;
14
15    // Print list
16    for (int i = 0; i < 3; i++)
17    {
18        printf("%i\n", list[i]);
19    }
20 }
```

```
1 // Implements a list of numbers with an array of dynamic size
2
3 #include <stdio.h>
4 #include <stdlib.h>
5
6 int main(void)
7 {
8     // List of size 3
9     int *list = malloc(3 * sizeof(int));
10    if (list == NULL)
11    {
12        return 1;
13    }
14
15    // Initialize list of size 3 with numbers
16    list[0] = 1;
17    list[1] = 2;
18    list[2] = 3;
19
20    // List of size 4
21    int *tmp = malloc(4 * sizeof(int));
22    if (tmp == NULL)
23    {
24        free(list);
25        return 1;
26    }
27
28    // Copy list of size 3 into list of size 4
29    for (int i = 0; i < 3; i++)
30    {
31        tmp[i] = list[i];
32    }
33
34    // Add number to list of size 4
35    tmp[3] = 4;
36
37    // Free list of size 3
38    free(list);
39
40    // Remember list of size 4
41    list = tmp;
42
```

```
43 // Print list
44 for (int i = 0; i < 4; i++)
45 {
46     printf("%i\n", list[i]);
47 }
48
49 // Free list
50 free(list);
51 return 0;
52 }
```

```
1 // Implements a list of numbers with an array of dynamic size using realloc
2
3 #include <stdio.h>
4 #include <stdlib.h>
5
6 int main(void)
7 {
8     // List of size 3
9     int *list = malloc(3 * sizeof(int));
10    if (list == NULL)
11    {
12        return 1;
13    }
14
15    // Initialize list of size 3 with numbers
16    list[0] = 1;
17    list[1] = 2;
18    list[2] = 3;
19
20    // Resize list to be of size 4
21    int *tmp = realloc(list, 4 * sizeof(int));
22    if (tmp == NULL)
23    {
24        free(list);
25        return 1;
26    }
27    list = tmp;
28
29    // Add number to list
30    list[3] = 4;
31
32    // Print list
33    for (int i = 0; i < 4; i++)
34    {
35        printf("%i\n", list[i]);
36    }
37
38    // Free list
39    free(list);
40    return 0;
41 }
```

```
1 // Implements a list of numbers with linked list
2
3 #include <stdio.h>
4 #include <stdlib.h>
5
6 // Represents a node
7 typedef struct node
8 {
9     int number;
10    struct node *next;
11 }
12 node;
13
14 int main(void)
15 {
16     // List of size 0
17     node *list = NULL;
18
19     // Add number to list
20     node *n = malloc(sizeof(node));
21     if (n == NULL)
22     {
23         return 1;
24     }
25     n->number = 1;
26     n->next = NULL;
27     list = n;
28
29     // Add number to list
30     n = malloc(sizeof(node));
31     if (n == NULL)
32     {
33         free(list);
34         return 1;
35     }
36     n->number = 2;
37     n->next = NULL;
38     list->next = n;
39
40     // Add number to list
41     n = malloc(sizeof(node));
42     if (n == NULL)
```

```
43 {
44     free(list->next);
45     free(list);
46     return 1;
47 }
48 n->number = 3;
49 n->next = NULL;
50 list->next->next = n;
51
52 // Print list
53 for (node *tmp = list; tmp != NULL; tmp = tmp->next)
54 {
55     printf("%i\n", tmp->number);
56 }
57
58 // Free list
59 while (list != NULL)
60 {
61     node *tmp = list->next;
62     free(list);
63     list = tmp;
64 }
65 return 0;
66 }
```

```
1 // Implements a list of numbers as a binary search tree
2
3 #include <stdio.h>
4 #include <stdlib.h>
5
6 // Represents a node
7 typedef struct node
8 {
9     int number;
10    struct node *left;
11    struct node *right;
12 }
13 node;
14
15 void free_tree(node *root);
16 void print_tree(node *root);
17
18 int main(void)
19 {
20     // Tree of size 0
21     node *tree = NULL;
22
23     // Add number to list
24     node *n = malloc(sizeof(node));
25     if (n == NULL)
26     {
27         return 1;
28     }
29     n->number = 2;
30     n->left = NULL;
31     n->right = NULL;
32     tree = n;
33
34     // Add number to list
35     n = malloc(sizeof(node));
36     if (n == NULL)
37     {
38         return 1;
39     }
40     n->number = 1;
41     n->left = NULL;
42     n->right = NULL;
```

```
43     tree->left = n;
44
45     // Add number to list
46     n = malloc(sizeof(node));
47     if (n == NULL)
48     {
49         return 1;
50     }
51     n->number = 3;
52     n->left = NULL;
53     n->right = NULL;
54     tree->right = n;
55
56     // Print tree
57     print_tree(tree);
58
59     // Free tree
60     free_tree(tree);
61 }
62
63 void free_tree(node *root)
64 {
65     if (root == NULL)
66     {
67         return;
68     }
69     free_tree(root->left);
70     free_tree(root->right);
71     free(root);
72 }
73
74 void print_tree(node *root)
75 {
76     if (root == NULL)
77     {
78         return;
79     }
80     print_tree(root->left);
81     printf("%i\n", root->number);
82     print_tree(root->right);
83 }
```