```c
// Implements a list of numbers with an array of fixed size

#include <stdio.h>

int main(void)
{
    // List of size 3
    int list[3];

    // Initialize list with numbers
    list[0] = 1;
    list[1] = 2;
    list[2] = 3;

    // Print list
    for (int i = 0; i < 3; i++)
    {
        printf("%i\n", list[i]);
    }
}
```

```c
// Implements a list of numbers with an array of dynamic size

#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    // List of size 3
    int *list = malloc(3 * sizeof(int));
    if (list == NULL)
    {
        return 1;
    }

    // Initialize list of size 3 with numbers
    list[0] = 1;
    list[1] = 2;
    list[2] = 3;

    // List of size 4
    int *tmp = malloc(4 * sizeof(int));
    if (tmp == NULL)
    {
        free(list);
        return 1;
    }

    // Copy list of size 3 into list of size 4
    for (int i = 0; i < 3; i++)
    {
        tmp[i] = list[i];
    }

    // Add number to list of size 4
    tmp[3] = 4;

    // Free list of size 3
    free(list);

    // Remember list of size 4
    list = tmp;

```

```c
    // Print list
    for (int i = 0; i < 4; i++)
    {
        printf("%i\n", list[i]);
    }

    // Free list
    free(list);
    return 0;
}
```

```c
// Implements a list of numbers with an array of dynamic size using realloc

#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    // List of size 3
    int *list = malloc(3 * sizeof(int));
    if (list == NULL)
    {
        return 1;
    }

    // Initialize list of size 3 with numbers
    list[0] = 1;
    list[1] = 2;
    list[2] = 3;

    // Resize list to be of size 4
    int *tmp = realloc(list, 4 * sizeof(int));
    if (tmp == NULL)
    {
        free(list);
        return 1;
    }
    list = tmp;

    // Add number to list
    list[3] = 4;

    // Print list
    for (int i = 0; i < 4; i++)
    {
        printf("%i\n", list[i]);
    }

    // Free list
    free(list);
    return 0;
}
```

```c
// Implements a list of numbers with linked list

#include <stdio.h>
#include <stdlib.h>

// Represents a node
typedef struct node
{
    int number;
    struct node *next;
}
node;

int main(void)
{
    // List of size 0
    node *list = NULL;

    // Add number to list
    node *n = malloc(sizeof(node));
    if (n == NULL)
    {
        return 1;
    }
    n->number = 1;
    n->next = NULL;
    list = n;

    // Add number to list
    n = malloc(sizeof(node));
    if (n == NULL)
    {
        free(list);
        return 1;
    }
    n->number = 2;
    n->next = NULL;
    list->next = n;

    // Add number to list
    n = malloc(sizeof(node));
    if (n == NULL)
```

```c
    {
        free(list->next);
        free(list);
        return 1;
    }
    n->number = 3;
    n->next = NULL;
    list->next->next = n;

    // Print list
    for (node *tmp = list; tmp != NULL; tmp = tmp->next)
    {
        printf("%i\n", tmp->number);
    }

    // Free list
    while (list != NULL)
    {
        node *tmp = list->next;
        free(list);
        list = tmp;
    }
    return 0;
}
```

```c
// Implements a list of numbers as a binary search tree

#include <stdio.h>
#include <stdlib.h>

// Represents a node
typedef struct node
{
    int number;
    struct node *left;
    struct node *right;
}
node;

void free_tree(node *root);
void print_tree(node *root);

int main(void)
{
    // Tree of size 0
    node *tree = NULL;

    // Add number to list
    node *n = malloc(sizeof(node));
    if (n == NULL)
    {
        return 1;
    }
    n->number = 2;
    n->left = NULL;
    n->right = NULL;
    tree = n;

    // Add number to list
    n = malloc(sizeof(node));
    if (n == NULL)
    {
        return 1;
    }
    n->number = 1;
    n->left = NULL;
    n->right = NULL;
```

```c
    tree->left = n;

    // Add number to list
    n = malloc(sizeof(node));
    if (n == NULL)
    {
        return 1;
    }
    n->number = 3;
    n->left = NULL;
    n->right = NULL;
    tree->right = n;

    // Print tree
    print_tree(tree);

    // Free tree
    free_tree(tree);
}

void free_tree(node *root)
{
    if (root == NULL)
    {
        return;
    }
    free_tree(root->left);
    free_tree(root->right);
    free(root);
}

void print_tree(node *root)
{
    if (root == NULL)
    {
        return;
    }
    print_tree(root->left);
    printf("%i\n", root->number);
    print_tree(root->right);
}
```