```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
// Prepends numbers to a linked list, using while loop to print

#include <cs50.h>
#include <stdio.h>
#include <stdlib.h>

typedef struct node
{
    int number;
    struct node *next;
}
node;

int main(int argc, char *argv[])
{
    // Memory for numbers
    node *list = NULL;

    // For each command-line argument
    for (int i = 1; i < argc; i++)
    {
        // Convert argument to int
        int number = atoi(argv[i]);

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

        // Prepend node to list
        n->next = list;
        list = n;
    }

    // Print numbers
    node *ptr = list;
    while (ptr != NULL)
    {
```

```c
        printf("%i\n", ptr->number);
        ptr = ptr->next;
    }

    // Free memory
    ptr = list;
    while (ptr != NULL)
    {
        node *next = ptr->next;
        free(ptr);
        ptr = next;
    }
}
```

```c
// Prepends numbers to a linked list, using for loop to print

#include <cs50.h>
#include <stdio.h>
#include <stdlib.h>

typedef struct node
{
    int number;
    struct node *next;
}
node;

int main(int argc, char *argv[])
{
    // Memory for numbers
    node *list = NULL;

    // For each command-line argument
    for (int i = 1; i < argc; i++)
    {
        // Convert argument to int
        int number = atoi(argv[i]);

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

        // Prepend node to list
        n->next = list;
        list = n;
    }

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

```c
    }

    // Free memory
    node *ptr = list;
    while (ptr != NULL)
    {
        node *next = ptr->next;
        free(ptr);
        ptr = next;
    }
}
```

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

#include <cs50.h>
#include <stdio.h>
#include <stdlib.h>

typedef struct node
{
    int number;
    struct node *next;
}
node;

int main(int argc, char *argv[])
{
    // Memory for numbers
    node *list = NULL;

    // For each command-line argument
    for (int i = 1; i < argc; i++)
    {
        // Convert argument to int
        int number = atoi(argv[i]);

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

        // If list is empty
        if (list == NULL)
        {
            // This node is the whole list
            list = n;
        }

        // If list has numbers already
        else
```

```
43          {
44              // Iterate over nodes in list
45              for (node *ptr = list; ptr != NULL; ptr = ptr->next)
46              {
47                  // If at end of list
48                  if (ptr->next == NULL)
49                  {
50                      // Append node
51                      ptr->next = n;
52                      break;
53                  }
54              }
55          }
56      }
57
58      // Print numbers
59      for (node *ptr = list; ptr != NULL; ptr = ptr->next)
60      {
61          printf("%i\n", ptr->number);
62      }
63
64      // Free memory
65      node *ptr = list;
66      while (ptr != NULL)
67      {
68          node *next = ptr->next;
69          free(ptr);
70          ptr = next;
71      }
72  }
```

```c
// Implements a sorted list of numbers using a linked list

#include <cs50.h>
#include <stdio.h>
#include <stdlib.h>

typedef struct node
{
    int number;
    struct node *next;
}
node;

int main(int argc, char *argv[])
{
    // Memory for numbers
    node *list = NULL;

    // For each command-line argument
    for (int i = 1; i < argc; i++)
    {
        // Convert argument to int
        int number = atoi(argv[i]);

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

        // If list is empty
        if (list == NULL)
        {
            list = n;
        }

        // If number belongs at beginning of list
        else if (n->number < list->number)
        {
```

```c
            n->next = list;
            list = n;
        }

        // If number belongs later in list
        else
        {
            // Iterate over nodes in list
            for (node *ptr = list; ptr != NULL; ptr = ptr->next)
            {
                // If at end of list
                if (ptr->next == NULL)
                {
                    // Append node
                    ptr->next = n;
                    break;
                }

                // If in middle of list
                if (n->number < ptr->next->number)
                {
                    n->next = ptr->next;
                    ptr->next = n;
                    break;
                }
            }
        }
    }

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

    // Free memory
    node *ptr = list;
    while (ptr != NULL)
    {
        node *next = ptr->next;
        free(ptr);
        ptr = next;
```

```
85        }
86    }
```

```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)
    {
        free_tree(tree);
        return 1;
    }
    n->number = 1;
    n->left = NULL;
```

```c
    n->right = NULL;
    tree->left = n;

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

    // Print tree
    print_tree(tree);

    // Free tree
    free_tree(tree);
    return 0;
}

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);
```

```
85      print_tree(root->right);
86  }
```