

- 1 ../README.txt
- 2
- 3 list*.c
- 4 tree.c

```
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 // Prepends numbers to a linked list, using while loop to print
2
3 #include <cs50.h>
4 #include <stdio.h>
5 #include <stdlib.h>
6
7 typedef struct node
8 {
9     int number;
10    struct node *next;
11 }
12 node;
13
14 int main(int argc, char *argv[])
15 {
16     // Memory for numbers
17     node *list = NULL;
18
19     // For each command-line argument
20     for (int i = 1; i < argc; i++)
21     {
22         // Convert argument to int
23         int number = atoi(argv[i]);
24
25         // Allocate node for number
26         node *n = malloc(sizeof(node));
27         if (n == NULL)
28         {
29             return 1;
30         }
31         n->number = number;
32         n->next = NULL;
33
34         // Prepend node to list
35         n->next = list;
36         list = n;
37     }
38
39     // Print numbers
40     node *ptr = list;
41     while (ptr != NULL)
42     {
```

```
43     printf("%i\n", ptr->number);
44     ptr = ptr->next;
45 }
46
47 // Free memory
48 ptr = list;
49 while (ptr != NULL)
50 {
51     node *next = ptr->next;
52     free(ptr);
53     ptr = next;
54 }
55 }
```

```
1 // Prepends numbers to a linked list, using for loop to print
2
3 #include <cs50.h>
4 #include <stdio.h>
5 #include <stdlib.h>
6
7 typedef struct node
8 {
9     int number;
10    struct node *next;
11 }
12 node;
13
14 int main(int argc, char *argv[])
15 {
16     // Memory for numbers
17     node *list = NULL;
18
19     // For each command-line argument
20     for (int i = 1; i < argc; i++)
21     {
22         // Convert argument to int
23         int number = atoi(argv[i]);
24
25         // Allocate node for number
26         node *n = malloc(sizeof(node));
27         if (n == NULL)
28         {
29             return 1;
30         }
31         n->number = number;
32         n->next = NULL;
33
34         // Prepend node to list
35         n->next = list;
36         list = n;
37     }
38
39     // Print numbers
40     for (node *ptr = list; ptr != NULL; ptr = ptr->next)
41     {
42         printf("%i\n", ptr->number);
```

```
43     }
44
45     // Free memory
46     node *ptr = list;
47     while (ptr != NULL)
48     {
49         node *next = ptr->next;
50         free(ptr);
51         ptr = next;
52     }
53 }
```

```
1 // Implements a list of numbers using a linked list
2
3 #include <cs50.h>
4 #include <stdio.h>
5 #include <stdlib.h>
6
7 typedef struct node
8 {
9     int number;
10    struct node *next;
11 }
12 node;
13
14 int main(int argc, char *argv[])
15 {
16     // Memory for numbers
17     node *list = NULL;
18
19     // For each command-line argument
20     for (int i = 1; i < argc; i++)
21     {
22         // Convert argument to int
23         int number = atoi(argv[i]);
24
25         // Allocate node for number
26         node *n = malloc(sizeof(node));
27         if (n == NULL)
28         {
29             return 1;
30         }
31         n->number = number;
32         n->next = NULL;
33
34         // If list is empty
35         if (list == NULL)
36         {
37             // This node is the whole list
38             list = n;
39         }
40
41         // If list has numbers already
42         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 }
```

```
1 // Implements a sorted list of numbers using a linked list
2
3 #include <cs50.h>
4 #include <stdio.h>
5 #include <stdlib.h>
6
7 typedef struct node
8 {
9     int number;
10    struct node *next;
11 }
12 node;
13
14 int main(int argc, char *argv[])
15 {
16     // Memory for numbers
17     node *list = NULL;
18
19     // For each command-line argument
20     for (int i = 1; i < argc; i++)
21     {
22         // Convert argument to int
23         int number = atoi(argv[i]);
24
25         // Allocate node for number
26         node *n = malloc(sizeof(node));
27         if (n == NULL)
28         {
29             return 1;
30         }
31         n->number = number;
32         n->next = NULL;
33
34         // If list is empty
35         if (list == NULL)
36         {
37             list = n;
38         }
39
40         // If number belongs at beginning of list
41         else if (n->number < list->number)
42         {
```

```
43     n->next = list;
44     list = n;
45 }
46
47 // If number belongs later in list
48 else
49 {
50     // Iterate over nodes in list
51     for (node *ptr = list; ptr != NULL; ptr = ptr->next)
52     {
53         // If at end of list
54         if (ptr->next == NULL)
55         {
56             // Append node
57             ptr->next = n;
58             break;
59         }
60
61         // If in middle of list
62         if (n->number < ptr->next->number)
63         {
64             n->next = ptr->next;
65             ptr->next = n;
66             break;
67         }
68     }
69 }
70 }
71
72 // Print numbers
73 for (node *ptr = list; ptr != NULL; ptr = ptr->next)
74 {
75     printf("%i\n", ptr->number);
76 }
77
78 // Free memory
79 node *ptr = list;
80 while (ptr != NULL)
81 {
82     node *next = ptr->next;
83     free(ptr);
84     ptr = next;
```

```
85     }  
86 }
```

```
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         free_tree(tree);
39         return 1;
40     }
41     n->number = 1;
42     n->left = NULL;
```

```
43     n->right = NULL;
44     tree->left = n;
45
46     // Add number to list
47     n = malloc(sizeof(node));
48     if (n == NULL)
49     {
50         free_tree(tree);
51         return 1;
52     }
53     n->number = 3;
54     n->left = NULL;
55     n->right = NULL;
56     tree->right = n;
57
58     // Print tree
59     print_tree(tree);
60
61     // Free tree
62     free_tree(tree);
63     return 0;
64 }
65
66 void free_tree(node *root)
67 {
68     if (root == NULL)
69     {
70         return;
71     }
72     free_tree(root->left);
73     free_tree(root->right);
74     free(root);
75 }
76
77 void print_tree(node *root)
78 {
79     if (root == NULL)
80     {
81         return;
82     }
83     print_tree(root->left);
84     printf("%i\n", root->number);
```

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