

```
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 // Start to build a linked list by prepending nodes
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 } node;
12
13 int main(void)
14 {
15     // Memory for numbers
16     node *list = NULL;
17
18     // Build list
19     for (int i = 0; i < 3; i++)
20     {
21         // Allocate node for number
22         node *n = malloc(sizeof(node));
23         if (n == NULL)
24         {
25             return 1;
26         }
27         n->number = get_int("Number: ");
28         n->next = NULL;
29
30         // Prepend node to list
31         n->next = list;
32         list = n;
33     }
34     return 0;
35 }
```

```
1 // Print nodes in a linked list with a while loop
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 } node;
12
13 int main(void)
14 {
15     // Memory for numbers
16     node *list = NULL;
17
18     // Build list
19     for (int i = 0; i < 3; i++)
20     {
21         // Allocate node for number
22         node *n = malloc(sizeof(node));
23         if (n == NULL)
24         {
25             return 1;
26         }
27         n->number = get_int("Number: ");
28         n->next = NULL;
29
30         // Prepend node to list
31         n->next = list;
32         list = n;
33     }
34
35     // Print numbers
36     node *ptr = list;
37     while (ptr != NULL)
38     {
39         printf("%i\n", ptr->number);
40         ptr = ptr->next;
41     }
```

```
42     return 0;  
43 }
```

```
1 // Print nodes in a linked list with a for loop
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 } node;
12
13 int main(void)
14 {
15     // Memory for numbers
16     node *list = NULL;
17
18     // Build list
19     for (int i = 0; i < 3; i++)
20     {
21         // Allocate node for number
22         node *n = malloc(sizeof(node));
23         if (n == NULL)
24         {
25             return 1;
26         }
27         n->number = get_int("Number: ");
28         n->next = NULL;
29
30         // Prepend node to list
31         n->next = list;
32         list = n;
33     }
34
35     // Print numbers
36     for (node *ptr = list; ptr != NULL; ptr = ptr->next)
37     {
38         printf("%i\n", ptr->number);
39     }
40     return 0;
41 }
```



```
1 // Free 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 } node;
12
13 int main(void)
14 {
15     // Memory for numbers
16     node *list = NULL;
17
18     // Build list
19     for (int i = 0; i < 3; i++)
20     {
21         // Allocate node for number
22         node *n = malloc(sizeof(node));
23         if (n == NULL)
24         {
25             return 1;
26         }
27         n->number = get_int("Number: ");
28         n->next = NULL;
29
30         // Prepend node to list
31         n->next = list;
32         list = n;
33     }
34
35     // Free memory
36     node *ptr = list;
37     while (ptr != NULL)
38     {
39         node *next = ptr->next;
40         free(ptr);
41         ptr = next;
42     }
```

```
43     return 0;  
44 }
```

```
1 // Appends numbers to a link 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 } node;
12
13 int main(void)
14 {
15     // Memory for numbers
16     node *list = NULL;
17
18     // Build list
19     for (int i = 0; i < 3; i++)
20     {
21         // Allocate node for number
22         node *n = malloc(sizeof(node));
23         if (n == NULL)
24         {
25             return 1;
26         }
27         n->number = get_int("Number: ");
28         n->next = NULL;
29
30         // If list is empty
31         if (list == NULL)
32         {
33             // This node is the whole list
34             list = n;
35         }
36
37         // If list has numbers already
38         else
39         {
40             // Iterate over nodes in list
41             for (node *ptr = list; ptr != NULL; ptr = ptr->next)
42             {
```

```
43         // If at end of list
44         if (ptr->next == NULL)
45         {
46             // Append node
47             ptr->next = n;
48             break;
49         }
50     }
51 }
52 }
53
54 // Print numbers
55 for (node *ptr = list; ptr != NULL; ptr = ptr->next)
56 {
57     printf("%i\n", ptr->number);
58 }
59
60 // Free memory
61 node *ptr = list;
62 while (ptr != NULL)
63 {
64     node *next = ptr->next;
65     free(ptr);
66     ptr = next;
67 }
68 return 0;
69 }
```

```
1 // Implements a sorted linked list of numbers
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 } node;
12
13 int main(void)
14 {
15     // Memory for numbers
16     node *list = NULL;
17
18     // Build list
19     for (int i = 0; i < 3; i++)
20     {
21         // Allocate node for number
22         node *n = malloc(sizeof(node));
23         if (n == NULL)
24         {
25             return 1;
26         }
27         n->number = get_int("Number: ");
28         n->next = NULL;
29
30         // If list is empty
31         if (list == NULL)
32         {
33             list = n;
34         }
35
36         // If number belongs at beginning of list
37         else if (n->number < list->number)
38         {
39             n->next = list;
40             list = n;
41         }
42     }
```

```
43     // If number belongs later in list
44     else
45     {
46         // Iterate over nodes in list
47         for (node *ptr = list; ptr != NULL; ptr = ptr->next)
48         {
49             // If at end of list
50             if (ptr->next == NULL)
51             {
52                 // Append node
53                 ptr->next = n;
54                 break;
55             }
56
57             // If in middle of list
58             if (n->number < ptr->next->number)
59             {
60                 n->next = ptr->next;
61                 ptr->next = n;
62                 break;
63             }
64         }
65     }
66 }
67
68 // Print numbers
69 for (node *ptr = list; ptr != NULL; ptr = ptr->next)
70 {
71     printf("%i\n", ptr->number);
72 }
73
74 // Free memory
75 node *ptr = list;
76 while (ptr != NULL)
77 {
78     node *next = ptr->next;
79     free(ptr);
80     ptr = next;
81 }
82 return 0;
83 }
```

```
1  // Frees memory in cases of error too
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 } node;
12
13 void unload(node *list);
14
15 int main(void)
16 {
17     // Memory for numbers
18     node *list = NULL;
19
20     // Build list
21     for (int i = 0; i < 3; i++)
22     {
23         // Allocate node for number
24         node *n = malloc(sizeof(node));
25         if (n == NULL)
26         {
27             unload(list);
28             return 1;
29         }
30         n->number = get_int("Number: ");
31         n->next = NULL;
32
33         // If list is empty
34         if (list == NULL)
35         {
36             list = n;
37         }
38
39         // If number belongs at beginning of list
40         else if (n->number < list->number)
41         {
42             n->next = list;
```

```
43         list = n;
44     }
45
46     // If number belongs later in list
47     else
48     {
49         // Iterate over nodes in list
50         for (node *ptr = list; ptr != NULL; ptr = ptr->next)
51         {
52             // If at end of list
53             if (ptr->next == NULL)
54             {
55                 // Append node
56                 ptr->next = n;
57                 break;
58             }
59
60             // If in middle of list
61             if (n->number < ptr->next->number)
62             {
63                 n->next = ptr->next;
64                 ptr->next = n;
65                 break;
66             }
67         }
68     }
69 }
70
71 // Print numbers
72 for (node *ptr = list; ptr != NULL; ptr = ptr->next)
73 {
74     printf("%i\n", ptr->number);
75 }
76
77 // Free memory
78 unload(list);
79 return 0;
80 }
81
82 void unload(node *list)
83 {
84     node *ptr = list;
```

```
85     while (ptr != NULL)
86     {
87         node *next = ptr->next;
88         free(ptr);
89         ptr = next;
90     }
91 }
```

```
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 }
```