```c
#include <stdio.h>
#include <stdlib.h>
#include "../include/ArrayList.h"

ArrayList* al_init(int mx_len)
{
    // Allocate space for list variable and initialize all attributes.
    ArrayList* new_list = (ArrayList*) malloc(sizeof(ArrayList));
    new_list->length = 0;
    new_list->max_length = mx_len;
    new_list->data = NULL;
    return new_list;
}

ArrayList* al_insert_end(ArrayList* list, TYPE new_val)
{
    if (list->length < list->max_length)
    {
        // Increase data array size by one and insert new element
        ++(list->length);
        list->data = (TYPE*) realloc(list->data, list->length * sizeof(TYPE));
        list->data[list->length - 1] = new_val;
    }
    else
        printf("al_insert_end: List is full.\n");
    return list;
}

ArrayList* al_insert_front(ArrayList* list, TYPE new_val)
{
    if (list->length < list->max_length)
    {
        // Increase data array size by one and insert new element
        ++(list->length);
        list->data = (TYPE*) realloc(list->data, list->length * sizeof(TYPE));

        int i = 0;

        // Shift existing elements to the right
        for (i = (list->length - 1); i > 0; --i)
            list->data[i] = list->data[i - 1];

        list->data[0] = new_val;
    }
    else
        printf("al_insert_front: List is full.\n");
    return list;
}

ArrayList* al_insert_at(ArrayList* list, TYPE new_val, int position)
{
    // position is 1-based
    if (position < 1 || position > list->length)
    {
        printf("al_insert_at: Invalid position.\n");
        return list;
    }

    if (list->length < list->max_length)
    {
        // Increase data array size by one and insert new element
        ++(list->length);
        list->data = (TYPE*) realloc(list->data, list->length * sizeof(TYPE));

        int i = 0;

        // Shift elements after position to the right
        for (i = (list->length - 1); i >= position; --i)
            list->data[i] = list->data[i - 1];
        list->data[position - 1] = new_val;
    }
    else
        printf("al_insert_at: List is full.\n");
```

```
 74     return list;
 75 }
 76 
 77 ArrayList* al_delete_front(ArrayList* list)
 78 {
 79     if (list->length > 0)
 80     {
 81         int i = 0;
 82 
 83         // Shift elements to the left
 84         for (i = 0; i < list->length; ++i)
 85             list->data[i] = list->data[i + 1];
 86 
 87         // Decrease size of data array to remove last element
 88         --(list->length);
 89         list->data = (TYPE*) realloc(list->data, list->length * sizeof(TYPE));
 90 
 91     }
 92     else
 93         printf("al_delete_front: List is empty.\n");
 94 
 95     return list;
 96 }
 97 
 98 ArrayList* al_delete_end(ArrayList* list)
 99 {
100     if (list->length > 0)
101     {
102         // Decrease size of data array to remove last element
103         --(list->length);
104         list->data = (TYPE*) realloc(list->data, list->length * sizeof(TYPE));
105     }
106     else
107         printf("al_delete_end: List is empty.\n");
108 
109     return list;
110 }
111 
112 ArrayList* al_delete_at(ArrayList* list, int position)
113 {
114     // position is 1-based
115     if (position < 1 || position > list->length)
116     {
117         printf("al_delete_at: Invalid position.\n");
118         return list;
119     }
120 
121     if (list->length > 0)
122     {
123         int i = 0;
124 
125         // Shift elements after position to the left
126         for (i = (position - 1); i < list->length; ++i)
127             list->data[i] = list->data[i + 1];
128 
129         // Decrease size of data array to remove last element
130         --(list->length);
131         list->data = (TYPE*) realloc(list->data, list->length * sizeof(TYPE));
132     }
133     else
134         printf("al_delete_at: List is empty.\n");
135 
136     return list;
137 }
138 
139 ArrayList* al_delete_with_val(ArrayList* list, TYPE val)
140 {
141     if (list->length > 0)
142     {
143         int i = 0;
144 
145         // Traverse till element with correct value is found
146         for (i = 0; i < list->length && list->data[i] != val; ++i)
147             ;
```

```
148
149         // Delete at the position where the traversal stopped in previous step
150         if (i >= list->length)
151             printf("al_delete_with_val: Element with value %d not found.\n",
152                     val);
153         else
154             list = al_delete_at(list, i + 1);
155     }
156     else
157         printf("al_delete_with_val: List is empty.\n");
158
159     return list;
160 }
161
162 ArrayList* al_reverse(ArrayList* list)
163 {
164     int i = 0, tmp = 0;
165
166     // Swap elements from the ends till the center is reached
167     for (i = 0; i < (list->length / 2); ++i)
168     {
169         tmp = list->data[i];
170         list->data[i] = list->data[list->length - 1 - i];
171         list->data[list->length - 1 - i] = tmp;
172     }
173     return list;
174 }
175
176 void al_display(ArrayList* list, char* msg)
177 {
178     printf("%s", msg);
179
180     if (list->length == 0)
181     {
182         printf("al_display: List is empty.\n");
183         return;
184     }
185
186     int i = 0;
187
188     for (i = 0; i < (list->length); ++i)
189         printf("%d ", list->data[i]);
190     printf("\n");
191 }
```