:2.1

```
static Node copyNode(Node node)
{
   if (node == NULL) {
      return NULL;
   }
   Node new_node = malloc(sizeof(*new_node));
   if (new_node == NULL) {
      return NULL;
   }
   new_node->x = node->x;
   new_node->next = NULL;
   return new_node;
}


static bool appendList(Node dest, Node src)
{
   if (dest == NULL) {
      return false;
   }
   while (dest->next != NULL) {
      dest = dest->next;
   }
   while (src != NULL) {
      Node new_node = copyNode(src);
      if (new_node == NULL) {
         return false;
      }
      dest->next = new_node;
      dest = dest->next;
      src = src->next;
   }
   return true;
}


static void destroyList(Node list)
{
   Node iterator = list;
   while (list != NULL) {
      iterator = iterator->next;
      free(list);
      list = iterator;
   }
}
```

```c
Node mergeSortedLists(Node list1, Node list2, ErrorCode* error_code)
{
   if (list1 == NULL || list2 == NULL || error_code == NULL) {
      if (error_code != NULL) {
         *error_code = NULL_ARGUMENT;
      }
      return NULL;
   }
   if (!isListSorted(list1) || !isListSorted(list2)) {
      *error_code = UNSORTED_LIST;
      return NULL;
   }
   Node head, iterator = NULL;
   while (list1 != NULL && list2 != NULL) {
      Node new_node;
      if (list1->x <= list2->x) {
         new_node = copyNode(list1);
         list1 = list1->next;
      }
      else {
         new_node = copyNode(list2);
         list2 = list2->next;
      }
      if (new_node == NULL) {
         destroyList(head);
         *error_code = MEMORY_ERROR;
         return NULL;
      }
      if (iterator == NULL) {
         head = new_node;
         iterator = new_node;
      }
      else {
         iterator->next = new_node;
         iterator = new_node;
      }
   }
   Node added_node = (list1 != NULL) ? list1 : list2;
   if (!appendList(iterator, added_node)) {
      destroyList(head);
      *error_code = MEMORY_ERROR;
      return NULL;
   }
   *error_code = SUCCESS;
   return head;
}
```

:2.2

```c
/*
 * 1. char* str2 declared not in the same place of usage
 * 2. int i declared not in the same place of usage
 * 3. char* str type is not const and str and x are not const pointers
 * 4. x pointing type should be size_t because of strlen return value
 * 5. no check if str is not NULL and x is not NULL
 * 6. changing x value and not pointed variable value (dangerous!!!)
 * 8. the allocation size is not correct and its better to use sizeof(char)
to tell the programmer what we are allocating (dangerous!!!)
 * 9. no check after malloc if the allocation succeeded (dangerous!!!)
 * 10. no null setting at the end of str2
 * 11. in str the indexing is incorrect (supposed to be str[*x - i - 1]
dangerous!!!)
 * 12. no braces after for (code convention)
 * 13. there is no reason to use both ifs if the other if checks the same
statement
 * 14. added new line to both of printfs
 * 15. x is not conventional name
 * 16. str2 is not conventional name
 * @ Working code issues:
 * 6. changing x value and not pointed variable value (dangerous!!!)
 * 8. the allocation size is not correct and its better to use sizeof(char)
to tell the programmer what we are allocating (dangerous!!!)
 * 11. in str the indexing is incorrect (supposed to be str[*x - i - 1]
dangerous!!!)
 * @ Code writing rules:
 * 1. char* str2 declared not in the same place of usage
 * 5. no check if str is not NULL and x is not NULL
 * 9. no check after malloc if the allocation succeeded (dangerous!!!)
 */

char* foo_updated(const char* const str, size_t* const str_length)
{
   if (!str_length || !str) {
      return NULL;
   }
   *str_length = strlen(str);

   char* reversed_str = malloc(sizeof(char) * (*str_length + 1));
   if (NULL == reversed_str) {
      return NULL;
   }
   reversed_str[*str_length] = '\0';

   for (int i = 0; i < *str_length; i++) {
      reversed_str[i] = str[*str_length - i - 1];
   }
   if (*str_length % 2 == 0) {
      printf("%s\n", str);
   }
   else {
      printf("%s\n", reversed_str);
   }
   return reversed_str;
}
```