CPS1011 - Programming Principles in C

Giorgio Grigolo

Semester 1 - 2021/22

Contents

Task 1 - Problem Solving

1.1 Header File

```c
1  #define MAX 200
2
3  char ERR_MSG[] = "\nERROR: No array is initialized.\n" ;
4
5  typedef struct {
6      int value;
7      int frequency;
8  } vf_pair_t;
9
10 typedef enum { false, true } boolean;
11
12 void print_menu(int *choice);
13
14 int init_array(int *array);
15
16 void reverse(const int *array_a, int *array_b, int length);
17
18 void display(int *array, int array_length);
19
20 void frequency(const int *array, vf_pair_t  *pairs, int array_length);
21
22 void display_frequencies(vf_pair_t *pairs, int array_length);
23
24 void jprintf(char *attribute, int value, boolean comma);
25
26 void clear_term();
```

In line 1, we define the absolute maximum of entries as required by the question. This (*and all the following*) is done in a header file to remove clutter from the main C source file.

A string for an error message is declared in line 2.

The type definition for a value-frequency pair containing two members of type `int` occurs in lines 5-7. An array of this type is later to be declared and used. Additionally, a type definition for the boolean data type is created to avoid importing the `<stdbool.h>` header file.

The remaining, are the pointers to the subroutines which are going to be defined and described in the following sections.

1.2 Function Definitions

1.2.1 `init_array()`

```c
 1 int init_array(int *input_array) {
 2     int length = 0;
 3 
 4     clear_term();
 5     printf("How many integers should this array hold? [1-200]\n> ");
 6 
 7     while (length <= 0 | length >= 200) {
 8         scanf("%d", &length);
 9 
10         if (length <= 0 | length > 200) {
11             printf("Invalid input. Try again.\n> ");
12         }
13     }
14 
15     for (int i = 0; i < length; i++) {
16         printf("Enter integer #%d\n> ", i + 1);
17         scanf("%d", &input_array[i]);
18     }
19 
20     clear_term();
21 
22     return length;
23 }
```

The above function iterates indefinitely, until a valid input is obtained for the length of the array to be initialized. Furthermore, it will iterate for as many times as was previously entered by the user, whilst requesting further input to populate the array specified in the function parameter `int *input_array`.

The `while` loop from lines 7-13 performs a **bitwise or** on the range check ensuring that the input is an integer such that $0 < \texttt{length} \leq 200$. An extremal bound check is performed in the `if` statement in lines 10-12 to alert the user about erroneous input in case the above restrictions are ignored.

A `for` loop is used in lines 15-18 to ask the user `length` times for an input which is stored in the location pointed to by the address stored in `input_array[i]`.

Finally, it returns the length specified by the user as an integer, to be later used in other functions.

1.2.2 display()

```c
1  void display(int *input_array, int array_length) {
2      clear_term();
3      printf("{\n%*s\"array\": [\n", 4, " ");
4      for (int i = 0; i < array_length; i++) {
5          printf("%*s{\n", 8, " ");
6          jprintf("offset", i, true);
7          jprintf("value", input_array[i], false);
8          printf("\n%*s", 8, " ");
9          if (i == array_length - 1)
10             printf("}\n");
11         else
12             printf("},\n\n");
13     }
14     printf("%*s]\n}\n", 4, " ");
15 }
```

The above function simply displays any integer array passed to it, specifically in the **JSON** format, given that its size is also passed to the function.

Firstly, the `clear_term()`[1] function is run to clear the terminal in preparation of the following output. In lines 3 and 14, the JSON header and footers are printed, which are independent from any data within. The notation `printf(%*s, n, "foo")` prints the string `"foo"` for `n` times consecutively. As such, it is used to have a fixed tab representation[2], regardless of environment or operating system.

Lines 4-13 contain a `for` loop which iterates through the array passed as `int *input_array` and prints a block of 4 lines with the help `jprintf()`[1] as follows:

1. 8 spaces followed by a `"{"`,
2. a JSON entry with attribute name `"offset"` and its value stored in `i`,
3. a JSON entry with attribute name `"value"` and its value stored in the address pointed to by `input_array[i]`
4. 8 spaces followed by a `"}"` and a `","` only if the current iteration is not the last one.

It is to be noted that each of these blocks represents one element in the `int *initial_array`.

[1]This will be analyzed later, in the **Utility Function Definitions** subsection.
[2]Tab representations may vary, and thus we ensure the convention of 1 tab = 4 spaces.

1.2.3 `reverse()`

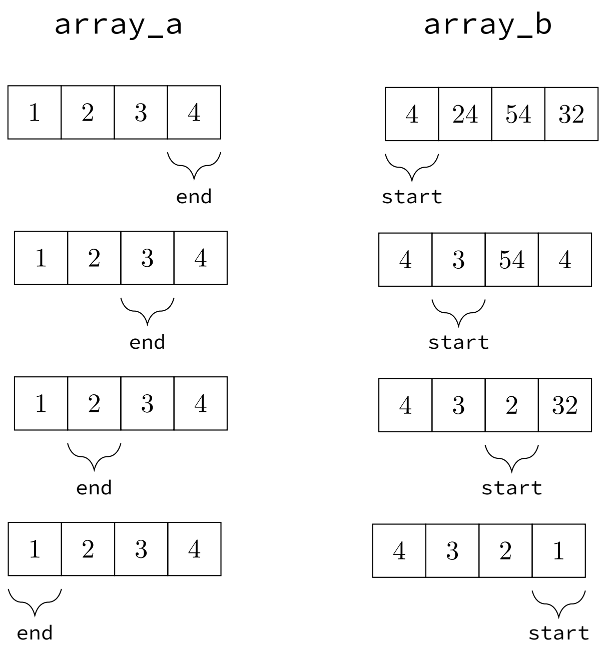
```c
1 void reverse(const int *array_a, int *array_b, int length) {
2     int start = 0;
3     int end = length - 1;
4     while (end >= 0) {
5         array_b[start] = array_a[end];
6         start++;
7         end--;
8     }
9 }
```

This function accepts two arrays containing pointers pointing to integers and their length. It is worth noting that since `array_a` is read-only, it has been declared as `const int *array_a`, i.e. constant.

In essence, the `while` loop that spans lines 4-8 iteratively sets the first element of `array_b` equal to the last element of `array_a` (line 5), then the second element of `array_b` equal to the second to last element of `array_b` and so on, until the counter `end` reaches 0.

Particularly, this is achieved by keeping track of the offsets for the two arrays stored inside the local variables `start` and `end` by incrementing the first, whilst decrementing the second.

Below is a block-view of an arbitrary run of the `reverse()` function, where `length` $= 4$.



1.2.4 frequency()

```c
void frequency(const int *array, vf_pair_t *pairs, int length) {
    for (int i = 0; i < length; i++) {
        pairs[i].frequency = -1;
    }
    for (int i = 0; i < length; i++) {
        int count = 1;
        for (int j = i + 1; j < length; j++) {
            if (array[i] == array[j]) {
                count++;
                pairs[j].frequency = 0;
            }
        }
        if (pairs[i].frequency != 0) {
            pairs[i].frequency = count;
            pairs[i].value = array[i];
        }
    }
}
```

1.2.5 display_frequencies()

```c
void display_frequencies(vf_pair_t *pairs, int array_length) {
    clear_term();
    printf("{\n%*s\"array\": [\n", 4, " ");
    int offset = 0;
    for (int i = 0; i < array_length; i++) {
        if (pairs[i].frequency != 0) {
            printf("%*s{\n", 8, " ");
            jprintf("offset", offset++, true);
            jprintf("value", pairs[i].value, true);
            jprintf("frequency", pairs[i].frequency, false);
            printf("\n%*s", 8, " ");
            if (i == array_length - 1)
                printf("}\n");
            else
                printf("},\n\n");
        }
    }
    printf("%*s]\n}\n", 4, " ");
}
```

1.3 Utility Function Definitions

1.3.1 `print_menu()`

```c
void print_menu(int *choice) {
    printf("1. Initialize the array\n2. Display the array\n3. Reverse a copy of the array and
     display it\n4. Display frequencies of entries in array\n5. Exit\n> ");
    scanf("%d", choice);
}
```

1.3.2 `clear_term()`

```c
void clear_term() {
    printf("\e[1;1H\e[2J");
}
```

1.3.3 `jprintf()`

```c
void clear_term() {
    printf("\e[1;1H\e[2J");
}
```

1.4 The `main()` Method

```c
int main() {

    int *array, *reversed_array, array_length;
    vf_pair_t *pairs;

    array = (int *) malloc(MAX * sizeof(int));
    reversed_array = (int *) malloc(MAX * sizeof(int));
    pairs = (vf_pair_t *) malloc(MAX * sizeof(vf_pair_t));

    if (array == NULL | reversed_array == NULL | pairs == NULL) {
        return 1;
    }

    boolean is_array_init = false;

    clear_term();

    int choice = 0;
    while (choice != 5) {
        print_menu(&choice);
        switch (choice) {
            case 1:
                array_length = init_array(array);
                is_array_init = true;
                break;
            case 2:
                is_array_init ?
                display(array, array_length)
                              : puts(ERR_MSG);
                break;
            case 3:
                is_array_init ?
                reverse(array, reversed_array, array_length),
                        display(reversed_array, array_length)
                              : puts(ERR_MSG);
                break;
            case 4:
                is_array_init ?
                frequency(array, pairs, array_length),
                        display_frequencies(pairs, array_length)
                              : puts(ERR_MSG);
                break;
            case 5:
                free(array);
                free(reversed_array);
                free(pairs);
                clear_term();
            default:
                break;
        }

    }
    return 0;
}
```

A DataTable library.