This course has already ended.

Fifth round tasks as a `ZIP file.`

Usually in the software projects there are situations where we are at the programming stage and cannot know how much memory the program needs in advance. In this case, we need a **dynamic memory** where size can be defined during the execution of the program. Dynamic memory is allocated by operating system upon request and generally it will always be free. This section gives introduction to computer memory organization, dynamic memory management and **valgrind** (tool is used to check memory leaks in the program).

Note: In this section, the tasks use valgrind tool to check memory leaks. If valgrind memory leaks or warnings are present in the code, only half of the total task point is given. If both valgrind and warnings exist, then you will get only quarter of the total task score. Valgrind information is provided later in this section.

Some memory management functions¶

The *string.h* header defines some functions for manipulating the memory content that may be useful in different situations. The main difference to the string functions discussed in section 4 is, that these functions do not care about the `"\0"` character that terminates strings, and therefore work with any data content.

- **memcpy** copies a block of memory to given address. The exact format is `void *memcpy(void *destination, const void *source, size_t n)`, where **source** points to the memory to be copied, **destination** points to where the block is to be copied, and **n** tells the number of bytes to be copied. The function returns pointer to the destination. Remember to ensure that the **destination** memory block is properly allocated, before writing to it, although neither source or destination memory needs to be located in heap: they could be local variables or arrays allocated from stack as well. The source and destination memory areas cannot overlap. If they do, memmove should be used instead.
- **memmove** is similar to memcpy, but also works when source and destination buffers overlap. The definition is also similar: `void *memmove(void *destination, const void *source, size_t n)`.
- **memcmp** compares two blocks of memory. The exact format is `int memcmp(const void *mem1, const void *mem2, size_t n)`, where **mem1** and **mem2** are the two memory blocks to be compared, and **n** is the size of the memory blocks in bytes. The function returns 0, if the two blocks are equivalent, or non-zero if they differ.
- **memset** fills the given memory block with given value. The exact form is `void *memset(void *mem, int c, size_t len)`. Here **mem** points to the block of memory to be modified, **len** is the size of the memory block in bytes, and **c** is the value that is set to every byte in the memory block. This function could be used, for example, to intialize an uninitialized memory block to zeros.

Here is an example that shows these functions in action:

```
#include <string.h> // memset, memcpy, memcmp, ...
#include <stdlib.h> // malloc, free
#include <stdio.h>  // printf

int main(void)
{
    char *mem1, *mem2;
    mem1 = malloc(1000);
    if (!mem1) {
        printf("Memory allocation failed\n");
        exit(-1);  // no use continuing this program
    }
    mem2 = malloc(1000);
    if (!mem2) {
        printf("Memory allocation failed\n");
        free(mem1);
        exit(-1);  // terminating
    }

    memset(mem1, 1, 1000);
    memset(mem2, 0, 1000);
    if (memcmp(mem1, mem2, 1000) != 0) {
        printf("the memory blocks differ\n");
    }
    memcpy(mem2, mem1, 1000);
    if (memcmp(mem1, mem2, 1000) == 0) {
        printf("the memory blocks are same\n");
    }
    free(mem1);
    free(mem2);
}
```

Task 5.4: Join Arrays¶

In this exercise, practice memory handling and then dynamic memory allocation

Implement the function **join_arrays** that gets three integer arrays and size of the three arrays as its arguments. The six function arguments should be in this order:

- number of integers in the first array (as unsigned integer)
- pointer to first array of integers
- number of integers in the second array (as unsigned integer)
- pointer to second array of integers
- number of integers in the third array (as unsigned integer)
- pointer to third array of integers

The function should join the three arrays into a single array that contains all integers from the original arrays in the above order. The new array should be allocated dynamically, and the function should return the pointer to the created array. You must not modify the original arrays.

Here's an example *main* - function, which can be used to test your program:

```
int main(void)
{
    /* testing exercise. Feel free to modify */
    int a1[] = { 1, 2, 3, 4, 5 };
    int a2[] = { 10, 11, 12, 13, 14, 15, 16, 17 };
    int a3[] = { 20, 21, 22 };

    int *joined = join_arrays(5, a1, 8, a2, 3, a3);

    for (int i = 0; i < 5 + 8 + 3; i++) {
        printf("%d  ", joined[i]);
    }
    printf("\n");

    return 0;
}
```

Points 20 / 20 My submissions 1
Deadline Friday, 9 July 2021, 19:59
To be submitted alone

This course has been archived (Saturday, 31 December 2022, 20:00).

Join Arrays

Select your files for grading

source.c

Choose File No file chosen

Submit

Task 5.5: Code Polisher¶

Implement function `char *delete_comments(char *input)` that removes C comments from program stored at *input*. *input* variable points to dynamically allocated memory. The function returns pointer to the polished program. You may allocate a new memory block for the output, or modify the content directly in the input buffer.

You'll need to process two types of comments:

- Traditional block comments delimited by `/*` and `*/`. These comments may span multiple lines. You should remove only characters starting from `/*` and ending to `*/` and for example leave any following newlines untouched.
- Line comments starting with `//` until the newline character. In this case, newline character must also be removed.

The function calling *delete_comments* only handles return pointer from *delete_ comments*. It does not allocate memory for any pointers. One way to implement *delete_comments* function is to allocate memory for destination string. However, if new memory is allocated then the original memory in *input* must be released after use.

The zip file linked to at the top of the page contains a testfile.c file for testing. You can also use other C-files.

Points 25 / 25 My submissions 4
Deadline Friday, 9 July 2021, 19:59
To be submitted alone

This course has been archived (Saturday, 31 December 2022, 20:00).

Polisher

Select your files for grading

source.c

Choose File No file chosen

Submit