Questions

1. What are the three parts of memory used by our C program? What are their main characteristics?

2. You are given a pointer **s** to the array of characters. How would you find its length?

3. Imagine that you've put two breakpoints in next program. One at return q line and another at main for loop line. Draw the memory content at those two points:

```
int main() {
        int * p = {1, 2, 3, 4, 5};
        int * q = reverse(p, 5);
        for (int i = 0; i < 5; i ++) {
                printf("%d, ", q[i]);
        }
}
int * reverse(int * p, int n) {
        static int q[n];
        int i = 0;
        for (; i < n; i++) {
                *(q + n - i - 1) = * (p + i);
        }
        return q;
}
```

4. What does *(q + n - i - 1) from the previous question mean?

5. What will be printed out when main function finishes?

6. Rewrite program from 3 so that new array is allocated on the heap? Make sure not to have memory leak.

7. Draw the memory content for the same breakpoints as in 3 for the program from 6.

8. Is the output of 6 same as the output of 3?

9. If I do sizeof(p) inside of the reverse function, what will be the result and why?

10. If i do sizeof(p) inside of the main function, what will be the result and why?

11. If I do sizeof(q) inside of the reverse function, what will be the result and why?

12. If I do sizeof(q) inside of the main function, what will be the result and why?

13. You have matrix mat with dimensions m and n. Write all possible prototypes of the function **f** which uses this matrix and returns void. Write all the ways you can access element in row i and column j.

14. You have to create new array inside of the function file that will be part of the library. How will you do it?

15. Transfer number 154 into binary, octal and hexadecimal if all numbers are stored in 1 byte of memory that is used for unsigned numbers.

16. Take the binary value from 14 and imagine that it stores signed value. What is the integer value of that number?

17. Transfer hexadecimal number ED into binary, octal and hexadecimal if all numbers are stored in 1 byte of memory that is used for unsigned numbers.

18. Transfer octal number 75 into binary, octal and hexadecimal if all numbers are stored in 1 byte of memory that is used for unsigned numbers.

19. How much is:
    a. 5 || 0
    b. 5 | 0
    c. 12 && 5
    d. 12 & 5
    e. ! 8
    f. ~ 8
20. Transfer numbers into binary value and show steps.
    a. -3.625
    b. 0.125

21. What is float value of 0? What is the float value of +inf/-inf?

22. What are the 4 steps when compiling C code and what is each step doing?

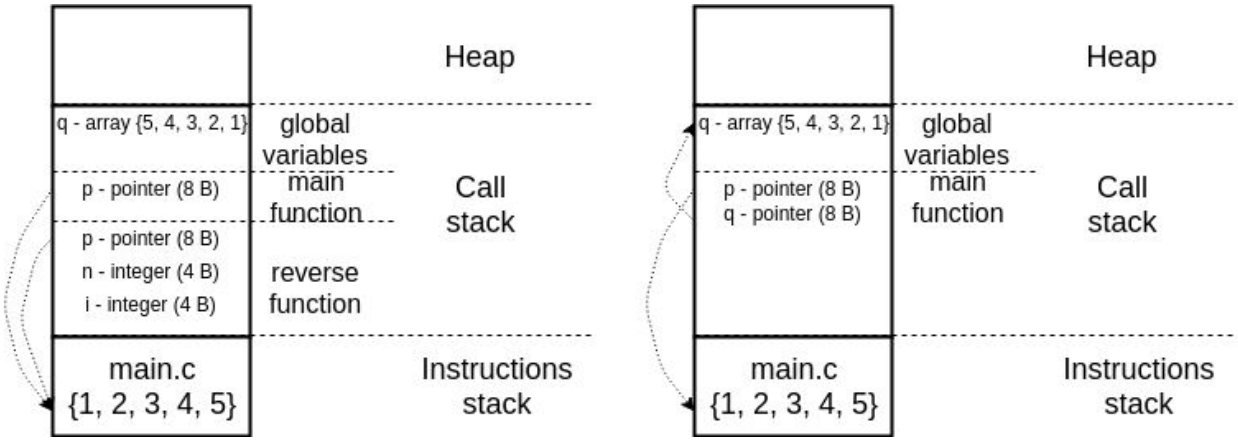
23. Create a data type **Worker** which is a structure that contains: **id**, pointer to Worker named **bosses**, pointer to Worker named **team_members**, pointer to char array named: **name**. Create and fill in the data for one Worker.

Answers

1. **Solution:**
    a. Code/instructions memory (read-only; contains the code, #define constants and string constants written in the code)
    b. Call stack (read and write; arguments and variables for each function separately are automatically written there; when function is exited all of its data is automatically deleted from the call stack)
    c. Heap (read and write; programer allocated and frees the space for variables and arrays)
2. **Solution:**

```
int i = 0;
for (; s[i] != '\0'; i++);
return i;
```

3. **Solution:**

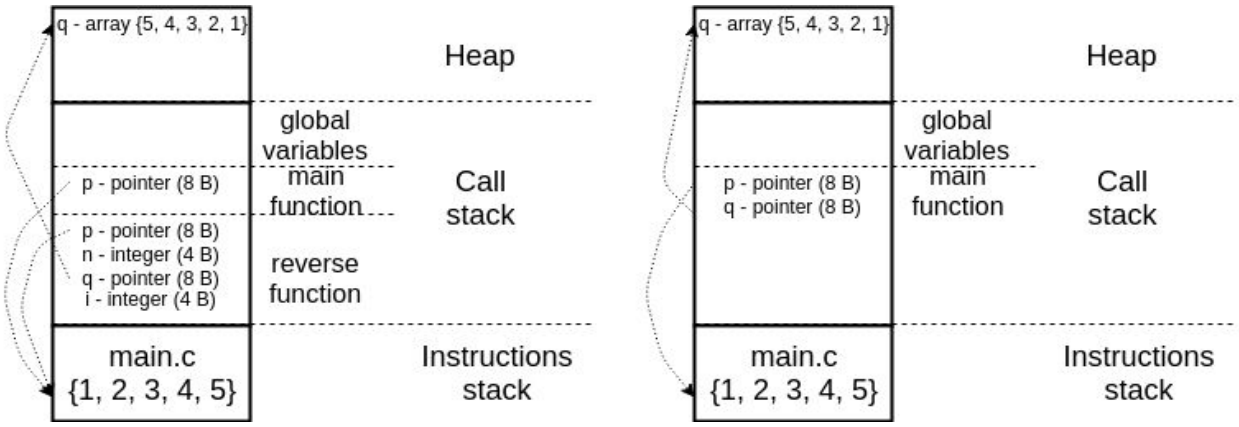


4. **Solution:** Same as q[n - i - 1] or basically i-th element from the end of the list
5. **Solution:** 5, 4, 3, 2, 1
6. **Solution:**

```
int main() {
        int * p = {1, 2, 3, 4, 5};
        int * q = reverse(p, 5);
        for (int i = 0; i < 5; i ++) {
                printf("%d\n", q[i]);
        }
        free(q);
        q = NULL;
}
int * reverse(int * p, int n) {
        int * q = malloc(n * sizeof(int));
        int i = 0;
        for (; i < n; i++) {
```

```
            *(q + n - i - 1) = * (p + i);
        }
        return q;
    }
```

7. **Solution:**



8. **Solution:** Yes
9. **Solution:** 8 bytes, p is a pointer to the constant array stored in the instructions part of memory; pointers always have size 8 bytes
10. **Solution:** 8 bytes, same as above
11. **Solution:** 20 bytes, q is a static array of 5 integer, each integer = 4 bytes => 5 * 4 = 20
12. **Solution:** 8 bytes, q is a pointer to a static array
13. **Solution**:
    a. void f (int **mat, int n, int m); //mat is a pointer to array of n pointers to the rows; // mat is defined as int ** mat; each row is defined as a pointer int * mat[i];
    b. void f (int n, int m, int (*mat)[m]); //mat is a pointer to array of n arrays of size m;
    c. void f (int n, int m, int mat[n][m]); //mat is a pointer to an array of n*m elements;
    d. void f (int *mat, int n, int m); //mat is a pointer to an array of n*m elements;
    e. # define N 10
       # define M 5
       void f (int **mat);
       void f (int (*mat)[M]);
       void f (int mat[N][M]);
       void f (int *mat);

    Element in i-th row and j-th column can be accessed in three ways:
    a. mat[i][j];
    b. *(mat + i * M + j)' // pass i rows of length M and j elements in the i-th row and then read the value
    c. *(mat[i] + j); //come to the i-th row and then move j elements in that row and read the value
14. **Solution:** using static array as in 3
15. **Solution:** binary:10011010; octal: 232; hexadecimal: 9A

16. **Solution:** -102, we get this by creating second complement of a = 10011010, which is ~a + 1 = 01100110 and then transforming that into a decimal value
17. **Solution:** binary:11101101; octal:355; decimal: 237
18. **Solution:** binary:00111101; decimal: 61; hexadecimal: 3D
19. **Solution:**
    a. 5 || 0 = 1 (result can be 1 or 0; result is 0 if both operands are 0, otherwise 1)
    b. 5 | 0 = 5 (bitwise OR 101 | 000 = 101, which is 5)
    c. 12 && 5 = 1 (result can be 1 or 0; if any operand is 0 results is 0, otherwise 1)
    d. 12 & 5 = 4 (bitwise AND 1100 & 0101 = 0100, which is 4)
    e. ! 8 = 0 (if value is different than zero, result is 0, otherwise result is 1)
    f. ~ 8 = 7 (bitwise NOT, ~1000 = 0111, which is 7)
20. **Solution**:
    a. 1 10000000 11010000000000000000

Number is negative, so first bit is 1. Next 8 bits are for exponent and last 23 bits are for mantissa. Split the number in the whole part and the fraction and convert them separately. 3_{10} => 11_2. 0.625 * 2 = 1.25; 0.25 * 2 = 0.5; 0.5 * 2 = 1.0, so binary fraction is 0.101 (read red numbers from left to right). $3.625_{10} = 11.101_2$. Now, we normalize that value, so that it starts with 1.. $11.101_2 = 1.1101_2 * 2^1$. Exponent of two is our exponent and we add bias 127 to it and transfer it to binary number. $1_{10} + 127_{10} = 128_{10} = 10000000_2$ - the exponent. Mantissa is fraction part padded with zeros to fit 23 bits, so 1101 and 19 zeros behind it.

    b. 0 01111100 00000000000000000000

Number is positive, so first bit is 0. There is no whole part, so we just work with fraction. 0.125 * 2 = 0.25; 0.25 * 2 = 0.5; 0.5 * 2 = 1.0, so binary fraction is 0.001 (read red numbers from left to right). $0.125_{10} = 0.001_2$. Now, we normalize that value, so that it starts with 1.. $0.001_2 = 1.0_2 * 2^{-3}$. Exponent of two is our exponent and we add bias 127 to it and transfer it to binary number. $-3_{10} + 127_{10} = 124_{10} = 01111100_2$ - the exponent. Mantissa is 0 and 22 zeros behind it.

21. **Solution**: Those are special values that can't be expressed with process above
    0 is 0 00000000 00000000000000000000000;
    +inf is 0 11111111 00000000000000000000000;
    -inf is 1 11111111 00000000000000000000000;
22. **Solution**:
    Preprocessor returns c code without where all # lines are resolved
    Compiler transfers c code into assembly language
    Assembler transfers assembly into machine code (0s and 1s)
    Linker links machine code from all the files together
23. typedef struct Worker Worker; // **struct Worker** is name of structure
    // and second **Worker** is name of data type defined with typedef
    struct Worker {long id; Worker * bosses; Worker * team_members; char * name;};
    Worker w;
    w.id = 50;
    w.bosses = malloc (2 * sizeof(Worker)); // creates an empty array for two bosses
    w.team_members = NULL; // doesn't have the team
    w.name = "Marija Stanojevic";