Example oral exam questions for EDAG01 Efficient C 2020

Jonas Skeppstedt

December 10, 2020

- No more example questions will be added to this file for the 2020 course.
- The questions are not organized in any way but added to the end as I create them so you can refer to their number e.g. in the discord channel.
- The purpose of the questions is to illustrate what kind of questions might be asked and should not be viewed as a complete set of questions for the oral exams.
- The most common question in the oral exam will be that I ask a follow-up question to your most recent reply, such as ”why is it so?”
- Above all, don’t be nervous for not passing the oral exam. If you have done the labs, the probability that you will pass is quite high.

1. What is the difference between `a` and `b` below? What can you do with them and when is the memory for the arrays deallocated?

```
int* f(int n)
{
        int*            a = calloc(n, sizeof(int));
        int             b[n];
}
```

2. Suppose you have a single-linked list that represents a set such as h in *intopt*, that the nodes have a field *next*, when h is empty, h is a null pointer, and you would like to write a function f to add a node q to the beginning of the list (so that the h in g below points to it), how would you call f in C (especially the arguments) and how could f be implemented? f should have two parameters h and p where p has type `node_t* p` and you should decide the type of the parameter h.

```
typedef struct node_t   node_t;

struct node_t {
        node_t*         next;
        /* other declarations. */
};

double g(void)
{
       node_t*         h;
       node_t*         q;

       /* more code. */

       /* call f somehow. */
}
```

3. As the previous question but you would like to put q at the end of the list instead.

4. Is this valid C and what does it mean?

```
struct {
        int     a : 1;
} s;
```

5. Why do C compilers use a stack pointer in the machine code they produce? Is it always used in every function or do some functions not need it?
6. Does pipelining reduce the number of clock cycles to execute an instruction, or what is the purpose of pipelining?
7. What is the purpose of a reorder buffer in a superscalar processor and what is it?
8. What is the purpose of rename registers in a superscalar processor?
9. What is the purpose of branch prediction in a superscalar processor?
10. What can a reasonable cache block size be?
11. What is meant by cache associativity and why is that useful?
12. Why are not fully associative data caches used?
13. Which two types of locality of references are exploited with caches to reduce the execution times of programs? Give examples of C code fragments in which each type of locality can be exploited.
14. What does `#undef` mean?
15. What does `#error` mean?
16. What does `#ifndef` mean?
17. What does `defined` mean? (yes, correct spelling — I do not refer to `#define`)
18. What does `##` mean?
19. What does `volatile` mean?
20. What does `restrict` mean?
21. What does `continue` mean?
22. What does `case` mean?
23. What does `default` mean?
24. What does designated initializer mean? Give an example?
25. What does implementation-defined behavior mean? Give an example.
26. What does unspecified behavior mean? Give an example.
27. What does undefined behavior mean? Give an example.
28. What does sequence point mean? Give an example.
29. What does static storage duration mean?
30. What does integer promotion mean?
31. What does the operator `? :` do?
32. How can you trigger a compile time error if your assumption about the size of an int does not hold? (hint: see previous question)

33. What does internal linkage mean?

34. What does the operator ~ do?

35. What does the operator & do?

36. What does the operator `|` do?

37. What does the operator >> do?

38. What does the operator << do?

39. What is the difference between `a = ++b` and `a = b++` ?

40. What is the return value of the function below?

```
#include <stdbool.h>
int f()
{
        bool   p = 1;

        return p + 1;
}
```

41. What is a compound literal?

42. What is the type of `1`?

43. What is the type of `1ULL`?

44. What is the type of `1.0`?

45. What what is wrong with `099`?

46. What does `alloca` do?

47. Can you add two pointers? If so, are there any restrictions?

48. Can you subtract two pointers? If so, are there any restrictions?

49. Can you add an integer and a pointer (`i+p`), and what would that mean?

50. Can you subtract an integer from a pointer (`p-i`), and what would that mean?

51. Can you subtract a pointer from an integer (`i-p`), and what would that mean?

52. What is dangerous with using a variable length array with a size n where n is read from input. Why is that not such a big problem when using `malloc`?

53. Why is `alloca` different (in addition to `alloca` not being ISO C) from using the VLA in the following loop?

```
void f(int n)
{
        int     i;
        int*    p;

        for (i = 0; i < n; i += 1) {
                int     a[i];
                p = alloca(i * sizeof(int));
                /* use a and p here... */
        }
}
```

54. Are there any restrictions on how `inline` may be used and if so why?

55. What do `setjmp` and `longjmp` do?

56. Which three of the following lines always work, which may crash, and why?

```
int main()
{
        char*   s = "hello";
        char    t[] = "hello";

        s[0] = ’H’;
        t[0] = ’H’;
}
```

57. Is returning a value `x` from `main` equivalent to calling `exit(x)` and why?

58. Can you copy a struct with an assignment statement like this?

```
struct { int a; } s, t;

void f()
{
        s = t;
}
```

59. Can you copy an array with an assignment statement like this?

```
int     a[10];
int     b[10];

void f()
{
        a = b;
}
```

60. What happens, if anything, to the elements of the array `a` in the statement in `f` ?

```
int     a[10];
int     b[10];
int*    p = a;
int*    q = b;

void f()
{
        p = q;
}
```

61. Why is the following code invalid C and would it, according to ISO C, help to use `const`?

```
int     n = 10;
int     a[n];

int main()
{
       return 0;
}
```

62. Why does the code below, which tries to read a number from stdin, not work?

```
#incluce <stdio.h>

#define ISDIGIT(c)      ((c) >= ’0’ && (c) <= ’9’)

int main()
{
        int     num;    // value of number.
        int     c;      // a char.

        num = 0;

        while (ISDIGIT(c = getchar()))
                num = 10 ∗ num + c - ’0’;

        printf("the number is %d\n", num);

        return 0;
}
```

63. Why is the following code invalid?

```
void f()
{
        int     a = 1;

        a = 2 ∗ ++a;
}
```

64. What is the value of `-1 / 2U > 4` ?

65. Is the following program valid and what does it print?

```
#include <stdio.h>

int main(void)
{
        signed int      a = 1;
        signed int∗     p = &a;
        unsigned int∗   q = (unsigned int∗)p;

        ∗q = 2;

        printf("a = %d\n", a);

        return 0;
}
```

Would there be a difference if `q` would be a pointer to a `signed short` and the cast be to that type? Motivate your answer.

66. Why is it a good idea to design a pipeline so that the different pipeline stages need approximately the same time to perform their work?

67. The profiler `operf` can make measurements on programs compiled without any special flag (such as `-pg` for `gprof`) how can it then know which function takes time? Not details but the basic principle.

68. Suppose you profile your program with `operf`, print the measurements with `opreport`, and find out that the function `pow` takes an unexpectedly large fraction of the execution time. How would you find out which other functions call `pow` and how many times?

69. How can you measure how many times each source code line is executed? Do you need to compile the program with some special flag and why in that case?

70. What can the Google sanitizer help you with?

71. What is `valgrind` and what can it do for you?

72. What is meant by *reduced* in RISC? Number of instructions or something else?

73. What is bad in the code below (by bad is meant risk for undefined behavior or other problem but not the fact that the code is quite meaningless since it does not do anything useful).

```
void f(size_t n)
{
        int*   p = calloc(n, sizeof(int));
        int*   q = p + 1;

        p = realloc(p, 2 * n * sizeof(int));

        if (p == NULL)
                return;

        q[0] = 1;
        p[n] += 2;

        free(p);
}
```

74. Why is the following code suboptimal and what can you do about it? How will your modification affect the program execution?

```
#define N (1000)

double a[N][N], b[N][N], c[N][N];

void matmul(void)
{
        size_t  i, j, k;

        for (i = 0; i < N; i += 1) {
                for (j = 0; j < N; j += 1) {
                        a[i][j] = 0;
                        for (k = 0; k < N; k += 1)
                                a[i][j] += b[i][k] * c[k][j];
                }
       }
}
```

75. Explain how the following function tests if the parameter a is a power of two.

    Assume $a > 0$, i.e., $a = 2^k$ for some $k \geq 0$.

```
int is_power_of_two(unsigned int a)
{
        return (a & (a-1)) == 0;
}
```

76. What is meant by SIMD vectorization? Give an example of what you should think of when you try to help the compiler with this.

77. In the code below, is it reasonable to expect that an optimizing ISO C compiler will produce machine code with only one multiplication? What would happen if you put back the assignment that is commented out? Would it affect your answer? Why or why not?

```
int     a;
int     b;
int     s;
float*  p;

int f()
{
        int   c;

        c = a * b;

        // *p = c;

        if (s > 1)
               c += a * b;
        return c;
}
```