Theory Tasks - Module 3, Satisfactory (S)

You should submit your answers as `pdf`, `jpg`, or `png` files.

The theory tasks in this exercise include content both from Module 2 and Module 3.

1 Binutils, Linkers, and Loaders

Please revisit Lecture 7, Part II about Linkers and Loaders. In the archive file `lecture-binutils.zip` (available on Canvas on page "Assignment Tasks and Workflow") there are many different examples. Take a closer look at the `Makefile` for different commands. For instance, `make hello-c` compiles file `hello-c.c` and disassembles the binary file using command `objdump`.

Go through the different examples, so that you understand the fundamentals of some of the most essential GNU binutils: `nm`, `objdump`, `size`, and `strip`. You can use the `man` command to get more info about the command, e.g., `man objdump`.

Tasks

In the first exercise of Module 2, you developed assembly code. In that project, there was a pre-existing file `factorial.c` that you compiled and used in your assembly project. You can find the original files in the folder `/template-projects/asm`. You can either copy these files from there again or use the files in your Module 2 solution.

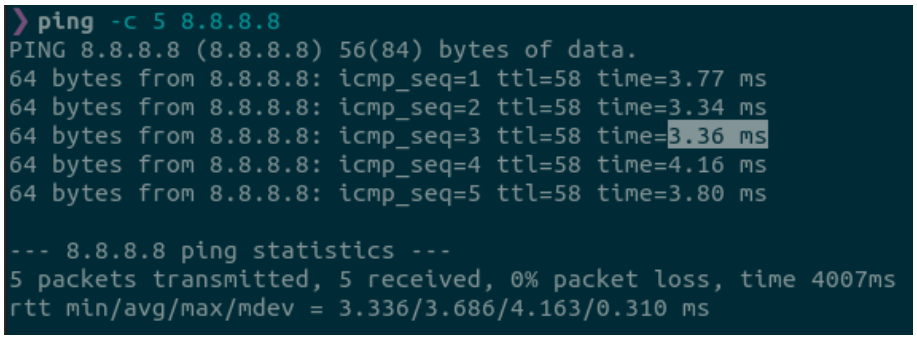
For each subtask, please provide a screenshot that:

1. Shows the command you executed.
2. Shows the relevant part of the output highlighted, with a bit of context around. If the output is short, you should include all of it. Otherwise, you may show an excerpt instead, as long as the context is still clear.

Note that you must also explain the meaning of each command you run in text.

For example, if the question was "If you ping 8.8.8.8, what's the latency of the third ping?" an answer could be formatted as follows:

> Ping takes an IP address to send messages to. The `-c` flag sets the number of messages to send, in this case to 5. The answer is 3.36ms.



Task 1a

Compile file `factorial.c` into an object file called `factorial.o`. The file should be compiled with maximal optimization (for performance) enabled, but still conforming to standard compliance (see the `man` page)

Task 1b

How large is the code in bytes of the compiled object file (with optimization enabled)? If you compile it again using no optimizations enabled, what is the size of the code then?

Task 1c

What is the size of the data section? Why is it not zero? In what way are these bytes used (inspect the C file and try to figure out an answer).

Task 1d

What is the size of the `bss` section? Explain why. When is the `bss` section used?

Task 1e

If you compile according to Step 1a, what is then the first instruction in function `fact` and the first instruction in function `factorial_message`?

Task 1f

Suppose you have compiled the main program using the command `gcc factorial.c main-c.c`. Give the command for listing all the symbols of the generated binary. What is the first symbol in the generated list? Also, as you can see, there are many more symbols than in the original C program. Where are all these symbols coming from?

Task 1g

Give the command that removes all symbols from a binary. Show how you can test that the symbols are removed. Explain why the symbols are not needed to execute the program, and why they are there, if they are not needed. Moreover, why *are* symbols indeed needed in an object file?

2 Garbage Collection

Go through Lecture 9 "Garbage Collection" again in detail. Specifically, make sure that you understand the details of mark-and-sweep collection, and that you have a good general knowledge of the other methods.

Task 2a

Suppose we have the following definition of a struct:

```
struct S {
    S * p1;
    S * p2;
};
```

Moreover, suppose that the following code is executed. Note that the code is divided into three separate parts:

```
  /* -- Part 1 -- */
  S* a = new S[1];
  S* b = new S[1];
  S* c = new S[1];
  S* d = new S[1];

  a[0].p1 = NULL;
  a[0].p2 = NULL;

  b[0].p1 = a;
  b[0].p2 = d;

  c[0].p1 = NULL;
  c[0].p2 = d;

  d[0].p1 = c;
  d[0].p2 = NULL;

  a = NULL;
  d = NULL;

  /* -- Part 2 -- */
  b = NULL;

  /* -- Part 3 -- */
  c = NULL;
```

In similar details as described at the lecture, draw and explain the layout (showing memory addresses, pointers, and arrows) of the memory, after executing Part 1.

Note that you need to add text of what is happening.

Task 2b

Suppose that Part 2 has been executed, and a mark-and-sweep garbage collection is invoked. Describe in detail the mark-and-sweep steps using this example. Clearly show what becomes garbage and why.

Task 2c

Again, explain what happens when mark-and-sweep GC executes after Part 3 has been executed.

Task 2d

After Part 3, what would happen if reference counting had been used instead? Clearly explain why that is the case.

Task 2e

Explain in your own words the difference between a copying garbage collector and a mark-and-sweep collector.

Task 2f

Explain in your own words why generational collectors and incremental collectors have some benefits. Discuss the main differences between them.

3 Register Allocation

Go through Lecture 11 "Liveness Analysis and Register Allocation" in detail. Specifically, make sure that you understand the details of liveness analysis, and that you have a good general knowledge about interference graphs and register allocation using graph coloring.

Task 3a

Consider the following Cigrid program:

```
1  int bar(int n) {
2    int i = 0;
3    int x = 2;
4
5    while (i < n) {
6      if (i > 10) {
7        x = x + n;
8        i = i + 2;
9      } else {
10       x = x * 3;
11       i++;
12     }
13   }
14   return x;
15 }
```

Do the following:

- Divide the program into basic blocks and draw the control-flow graph of the program. Make sure to include the program statements in the blocks.
- For all basic blocks, clearly write out *def* and *use* sets.
- Illustrate how the liveness analysis algorithm works for this example. Clearly show each iteration, and how *live-in* and *live-out* sets are computed in each iteration.
- Show the end result when the algorithm has reached a fixed point.

Task 3b

Construct the interference graph for the control-flow graph above.

Task 3c

Informally explain how an interference graph can be used when performing register allocation using graph coloring. You do not need to show it with the example above. Instead, describe clearly the intuition of how the interference graph is used in register allocation.

4 Dominators

Go through Lecture 13 and learn about the different aspects of dominators.

Task 4a

Suppose you have a directed graph $G = (V, E)$ with a start node 1 that is part of the set V. The graph is defined as follows.

$$V = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\} \tag{1}$$

$$\begin{aligned} E = \{&(1,2),(2,3),(3,2),(2,4),(3,5),(4,5),(5,11), \\ &(1,11),(1,6),(6,7),(7,10),(6,8),(8,10), \\ &(6,9),(9,10),(10,6),(10,11),(11,12)\} \end{aligned} \tag{2}$$

Clearly draw the graph and mark each vertex with the given vertex number.

Task 4b

Which are the dominator sets for the following vertices: $\{3, 5, 9, 10, 12\}$?

Task 4c

Which are the immediate dominators for vertices 10 and 11?

Task 4d

Write down the dominator tree for the graph.

Task 4e

Is vertex 4 or vertex 6 a header of a loop (see Lecture 13)? Give a detailed answer based on the formal definition of a loop. If it is a loop, state the set S.

Task 4f

Explain in your own words what hoisting means. Give an example that illustrates hoisting and loop invariants. Explain the example in detail.

5 Static Single Assignment (SSA)

Go through Lecture 14 in detail.

Task 5a

Give a clear explanation (in your own words) what SSA is and why it is useful. Explain in detail using an example. You need to explain the idea of variable renaming and phi-functions. You should invent your own example and not use one of the examples in the lectures. Please show your example both when it is in SSA form and when it is not.

Task 5b

Consider again the graph from Task 4a. What are the dominance frontier sets for the following vertices: $\{2, 3, 10, 11\}$?

Task 5c

Explain in your own words how an automatic conversion into SSA can be done. You do not have to give an algorithm or use formal notation, but you should provide examples and a description that clearly shows the main ideas.

Task 5d

Dead code elimination can easily be done if the program is in SSA form. Explain why it is "easier" than if the program was NOT in SSA form. In what way would the algorithm have to be different if it was not in SSA form?