ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE
EIDGENÖSSISCHE TECHNISCHE HOCHSCHULE – LAUSANNE
POLITECNICO FEDERALE – LOSANNA
SWISS FEDERAL INSTITUTE OF TECHNOLOGY – LAUSANNE

Faculté Informatique et Communications
CS–202 Computer Systems
Argyraki K., Bugnion E. & Chappelier J.-C.

CS–202 COMPUTER SYSTEMS

Final Exam solution

June 17th, 2024

INSTRUCTIONS (please read carefully)

IMPORTANT! Please strictly follow these instructions, otherwise your exam may be canceled.

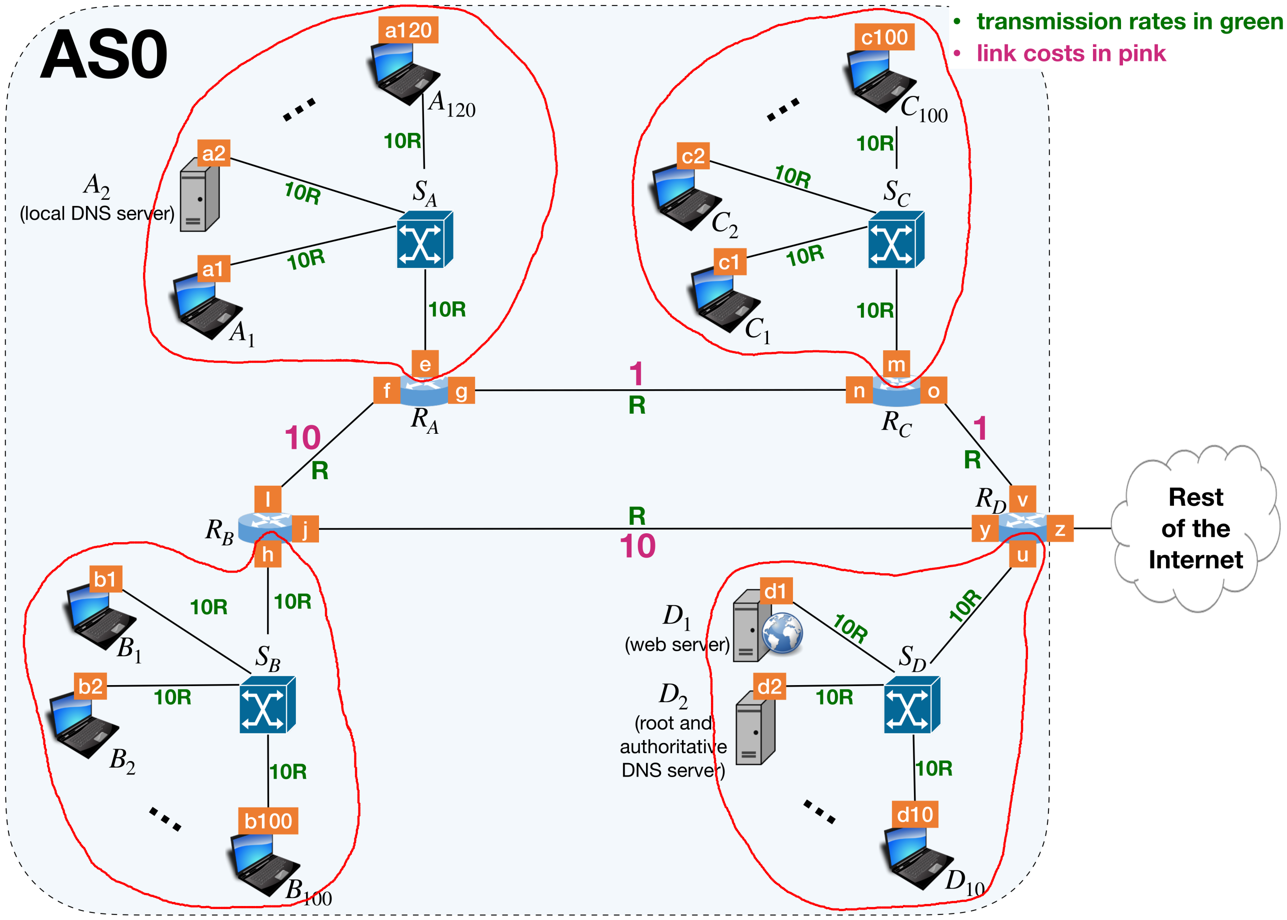
1. You have three hours to complete this examination (3:15–6:15pm).
2. You must **use black or dark blue ink**, neither pencil nor any other color.
3. This is closed book exam.
   Personal notes, two times dual-sided A4 sheets (4 sides in total), allowed.

   On the other hand, you may not use any personal computer, mobile phone or any other electronic equipment.
4. Answer the questions directly on the exam sheet; do not attach any any additional sheets; only this document will be graded.
5. Carefully and *completely* read each question so as to do only what we actually ask for. If the statement seems unclear, or if you are in any doubt, ask one of the assistants for clarification.
6. The exam consists of six exercises, which can be addressed in any order, except Question 2 which is a continuation of Question 1. These exercises do not score the same; points are indicated; the total is 125 points. All exercises count for the final grade.

LEAVE THIS EMPTY						
Question 1	Question 2	Question 3	Question 4	Question 5	Question 6	**TOTAL**
25	30	19	12	14	25	125

Question 1 – Subnets, prefixes, and packets [25 points]

Consider the topology in Figure 1, which shows a single Autonomous System (AS), AS0.



(For convenience, a copy is also provided on draft paper.)

Figure 1: Network topology for Questions 4 and 5.

- There are 4 routers, R_A, R_B, R_C, and R_D. Router R_D is the only border router of AS0.
- There are 4 Ethernet switches, S_A, S_B, S_C, and S_D. These switches do not have IP addresses.
- There are 120 end-systems A_1 to A_{120}; 100 end-systems B_1 to B_{100}; 100 end-systems C_1 to C_{100}; and 10 end-systems D_1 to D_{10}.
- The pink numbers represent the costs of links between the routers.
- The green numbers represent the transmission rates of links.
- A_2 and D_2 are DNS servers. D_2 is both a root DNS server, and an authoritative DNS server for `epfl.ch`. A_2 is neither root, nor authoritative DNS server for any domain.
- A_1 uses A_2 as its local DNS server.
- Whenever A_2 needs to contact a root DNS server, it contacts D_2.
- D_1 is a web server with DNS name `d1.epfl.ch`.
- The small orange boxes are network interfaces. E.g., router R_A has network interfaces e, f, and g.

① **[1 point]** Identify all the IP subnets inside AS0 by **marking them on Figure 1 on the left** (not on the provided extra colored sheet).

② **[10 points]** The administrator of this AS owns the following IP prefixes:

- 5.0.0.0/24
- 5.0.1.0/24

From these two IP prefixes, assign an IP prefix to each IP subnet that contains end-systems. Each IP prefix you assign must have the smallest possible size. **Justify your answer.**

Answer and justification:

IP subnet A:

- 122 IP addresses: 120 end-systems, 1 router interface, 1 broadcast IP address.
- We need 7 bits.
- 5.0.0.0 = 00000101.00000000.00000000.00000000.
- We can assign IP prefix 00000101.00000000.00000000.0xxxxxxx = 5.0.0.0/25.

IP subnet B:

- 102 IP addresses: 100 end-systems, 1 router interface, 1 broadcast IP address.
- We need 7 bits.
- We flip one bit from the previously allocated prefix: 00000101.00000000.00000000.1xxxxxxx = 5.0.0.127/25.

At this point, we have exhausted IP prefix 5.0.0.0/24, and we need to start allocating from the other one.

IP subnet C:

- 102 IP addresses: 100 end-systems, 1 router interface, 1 broadcast IP address.
- We need 7 bits.
- 5.0.1.0 = 00000101.00000000.00000001.00000000.
- We can assign IP prefix 00000101.00000000.00000001.0xxxxxxx = 5.0.1.0/25.

IP subnet D:

- 12 IP addresses: 10 end-systems, 1 router interface, 1 broadcast IP address.
- We need 4 bits.
- We flip one bit from the previously allocated prefix: 00000101.00000000.00000001.1xxxxxxx.
- We keep only 4 bits: 00000101.00000000.00000001.1000xxxx = 5.0.1.127/28.

③ **[2 points]** For each of the 4 routers, state how many routing protocol it participates in. **Justify your answer.**

Answer and justification:

Routers R_A, R_B, R_C, and R_D participate in AS0's intra-AS routing protocol, through which they exchange routes to the AS's local IP subnets.

Moreover, router R_D participates in the inter-AS routing protocol (BGP), through which it learns routes to foreign ASes and also advertizes to foreign ASes a route to AS0.

④ **[2 points]** How many and which IP prefix(es) do you expect border router R_D to advertize to other ASes? **Justify your answer.**

Answer and justification:

Router R_D should advertize one or more IP prefixes that contain exactly its entire address space (the two IP prefixes it owns): no less and no more.

The smallest IP prefix that contains exactly 5.0.0.0/24 and 5.0.1.0/24 is 5.0.0.0/23, so we expect R_D to advertize this IP prefix.

⑤ **[10 points]** All end-systems and packet switches have been rebooted. All caches (of all kinds) are empty. The user of end-system A_1 types in their web browser `http://d1.epfl.ch/image.png`. This is a large image (it does not reference any other object).

List the sequence of packets that are received or forwarded (sent) by network interface e of router R_A as a result of the end-user's action, up to and including the first packet that carries D_1's HTTP response. You do not need to list the packets received or forwarded by other network interfaces of router R_A.

Answer by completing Table 1 on the next page. The first row shows an example (which is not part of the correct answer). You may not need to fill all the rows of the table. **If you need to make any assumptions, state them.**

Answer:

#	Source MAC	Dest MAC	Source IP	Dst IP	Transp. prot.	Src Port	Dst Port	Application & Purpose
Example	x	y	x	y	UDP	5000	6000	Request for file ...
1	a_1	broadcast	-	-	-	-	-	ARP request for a_2's MAC address
2	a_2	broadcast	-	-	-	-	-	ARP request for e's MAC address
3	e	a_2	-	-	-	-	-	ARP response with e's MAC address
4	a_2	e	a_2	d_2	UDP	1000	53	DNS request for d_1's IP address
5	e	a_2	d_2	a_2	UDP	53	1000	DNS response with d_1's IP address
6	a_1	broadcast	-	-	-	-	-	ARP request for e's MAC address
7	e	a_1	-	-	-	-	-	ARP response with e's MAC address
8	a_1	e	a_1	d_1	TCP	2000	80	Connection setup request (SYN)
9	e	a_1	d_1	a_1	TCP	80	2000	Connection setup response (SYN ACK)
10	a_1	e	a_1	d_1	TCP	2000	80	HTTP GET request for `image.png`
11	e	a_1	d_1	a_1	TCP	80	2000	HTTP OK response, first segment

Table 1: Packets received or sent by network interface e of router R_A.

Question 2 – TCP and delay computation [30 points]

Consider the same network topology as in Question 1 (Figure 1, page 2) and the events of Question 1⑤. Assume that:

- Transport-layer, network-layer, and link-layer headers have insignificant size.
- A_1's receiver window is always 5 000 bytes.
- The links between the 4 routers have transmission rate R, in both directions.
- All the other links have transmission rate 10R, in both directions.
- All links have propagation delay D, in both directions.
- All network devices are store-and-forward (as we saw in class) and have infinite queues.

Recall that, in Question 1⑤, A_1 makes an HTTP request to D_1 and receives an HTTP response. A_1 sends no data to D_1 other than this HTTP request. No segment is lost or reordered during the entire exchange.

① **[5 points]** Assume the following sizes:

- Maximum Segment Size (MSS): 1 byte.
- HTTP request: 1 byte.
- HTTP response, including HTTP header and `image.png`: 12 bytes.

The diagram in Figure 2 shows the beginning of the communication between A_1 and D_1. The next sequence number (SEQ) that A_1 is expecting after the 3-way handshake is SEQ 1. Complete the diagram by showing:

- All the segments (including the segments that carry only ACKs) exchanged until A_1 receives the entire image.
- The SEQ numbers of D_1's segments.
- The ACK numbers of A_1's segments.
- The status of D_1's congestion-control algorithm.
- The values of D_1's congestion window and ssthresh.

Answer **here**: (do NOT answer on the provided DRAFT copy)

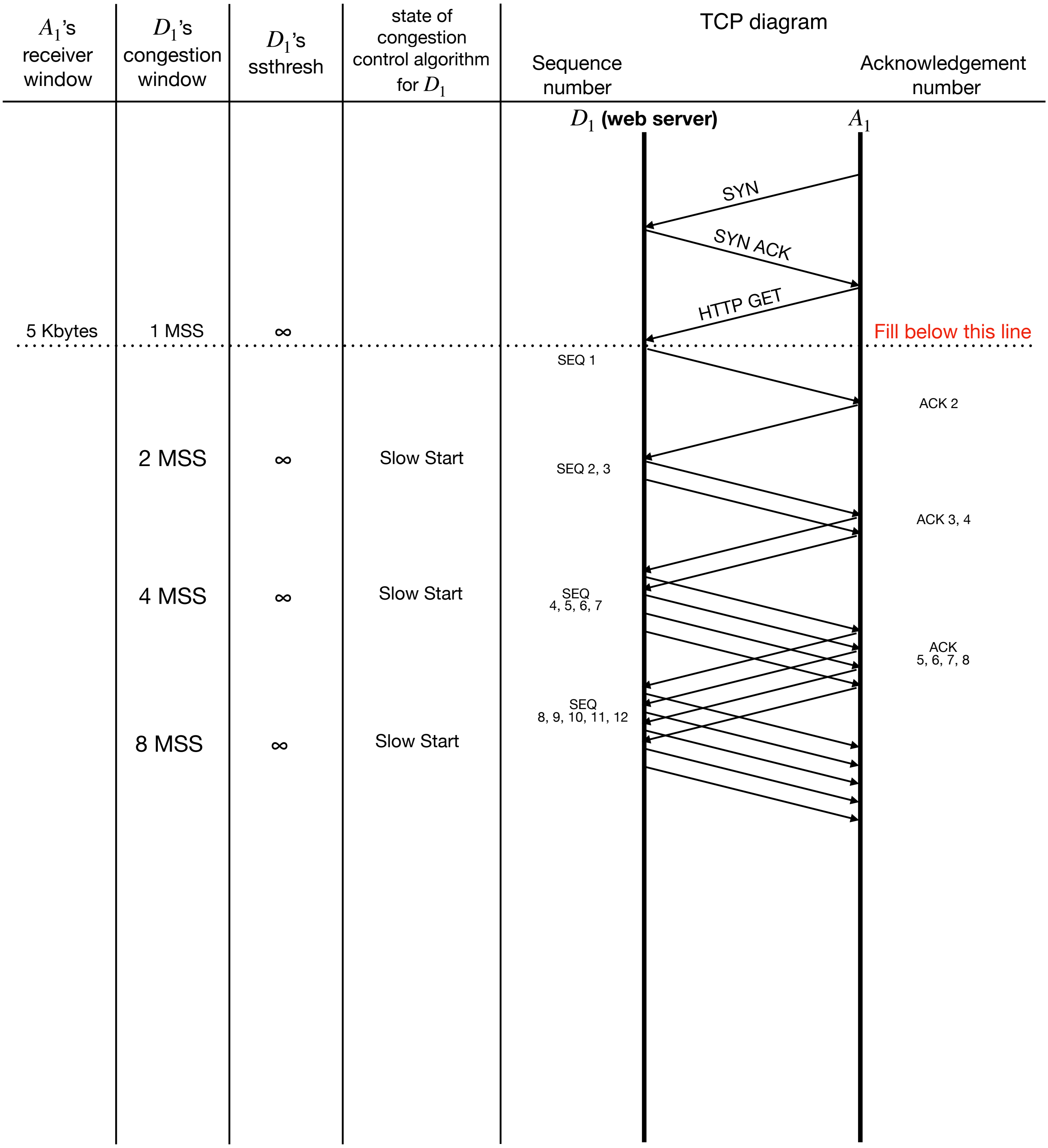


Figure 2: TCP diagram for Question 5.

continues on back ☞

② **[6 points]** Now assume the following sizes:

- MSS: 1000 bytes.
- HTTP request: 1000 bytes.
- HTTP response, including HTTP header and `image.png`: 12 000 bytes.

Does the TCP diagram change? If yes, in what way(s)? (You don't have to redraw the entire diagram from scratch, just state or sketch the changes, if any.) Is it flow control, or congestion control that determines the rate at which D_1 sends in this particular scenario? **Justify your answer.**

Answer and justification:

The first thing that changes is the sequence and acknowledgment numbers: they both increase by 1000 instead of 1. E.g., D_1 sends SEQ 1, then SEQ 1001, SEQ 2001, etc; while A_1 sends ACK 1001, ACK 2001, ACK 3001, etc.

The second thing that changes is D_1's sender window: When D_1 receives ACK 5001, it sets its congestion window to 5000, which is equal to the receiver window. From that point on, the congestion window continues to increase normally (by 1 MSS with every new ACK), however, the sender window remains at 5000 bytes.

However, this does not effectively change the diagram, because a sender window of 5000 bytes is enough for D_1 to send all the remaining bytes of the HTTP response during the fourth round.

At first it is congestion control that determines the rate at which D_1 sends, because D_1's congestion window is smaller than A_1's receiver window. As soon as D_1's congestion window reaches 5000 bytes, from that point on, it is flow control, because the receiver window is smaller. However, in practice, that does not make any difference, because the receiver window is large enough for D_1 to send all the remaining bytes during the fourth round.

③ **[4 points]** Assume the same parameters as in sub-question ②.

Consider D_1's second segment, i.e., the first segment that carries (the first part of) D_1's HTTP response to A_1. How long does it take from the moment D_1 transmits the first bit of this segment until A_1 receives the last bit of this segment? **Justify your answer.**

Answer and justification:

- One transmission-delay component per link: $\frac{1000}{10R} + \frac{1000}{10R} + \frac{1000}{R} + \frac{1000}{R} + \frac{1000}{10R} + \frac{1000}{10R} = \frac{2400}{R}$.
- One propagation-delay component per link: $6D$.
- Total: $\frac{2400}{R} + 6D$.

④ **[10 points]** Assume the same parameters as in sub-question ②.

Now consider the entire sequence of D_1's segments that carry D_1's HTTP response to A_1. How long does it take from the moment D_1 transmits the first bit of the first segment until A_1 receives the last bit of the last segment? **Justify your answer.**

Be careful: D_1 and A_1 are not connected by a single link (as was the case in some of the practice exercises).

Answer and justification:

Looking at the TCP diagram, the total delay consists of:

- Three times the time for one data segment to go from D_1 to A_1 plus one ACK segment to go from A_1 to D_1.
- The time for the last 5 segments to go from D_1 to A_1.

First component:

- Data segment from D_1 to A_1: $\frac{2400}{R} + 6D$.
- ACK segment from A_1 to D_1: $6D$.
- Total: $3 \times \left(\frac{2400}{R} + 12D\right) = \frac{7200}{R} + 36D$.

Second component:

- Delay of first packet to bottleneck: $\frac{1000}{10R} + \frac{1000}{10R} + 2D = \frac{200}{R}$.
- Delay of all packets on bottleneck: $\frac{5000}{R} + D$.
- Delay of last packet after bottleneck: $\frac{1000}{R} + \frac{1000}{10R} + \frac{1000}{10R} + 3D = \frac{1200}{R} + 3D$.
- Total: $\frac{6400}{R} + 6D$.

The two together: $\frac{13600}{R} + 42D$.

⑤ **[5 points]** Which is the maximum number of segments that may be lost and not affect at all the answer to sub-question ④? If you need to assume a timeout value, assume that the timeout value is fixed at 4RTT. **Justify your answer.**

Answer and justification:

No data-carrying segment may be lost, otherwise there would be a timeout (or a fast retransmit) and that would change the TCP diagram (hence the delay for the HTTP response to reach A_1).

One segment that carries only an ACK may be lost without changing the TCP diagram: either ACK 5001 or ACK 6001. Figure 3 shows these two scenarios.

Losing one of these two ACKs does not change the TCP diagram because:

- TCP ACKs are cumulative. Hence, as long as D_1 receives ACK 7001, it knows that A_1 received all the data segments up to and including SEQ 6001.
- Losing one of these ACKs causes D_1 to advance its sender window more slowly, but it still transmits the last 5 data segments at the maxiumum possible rate.

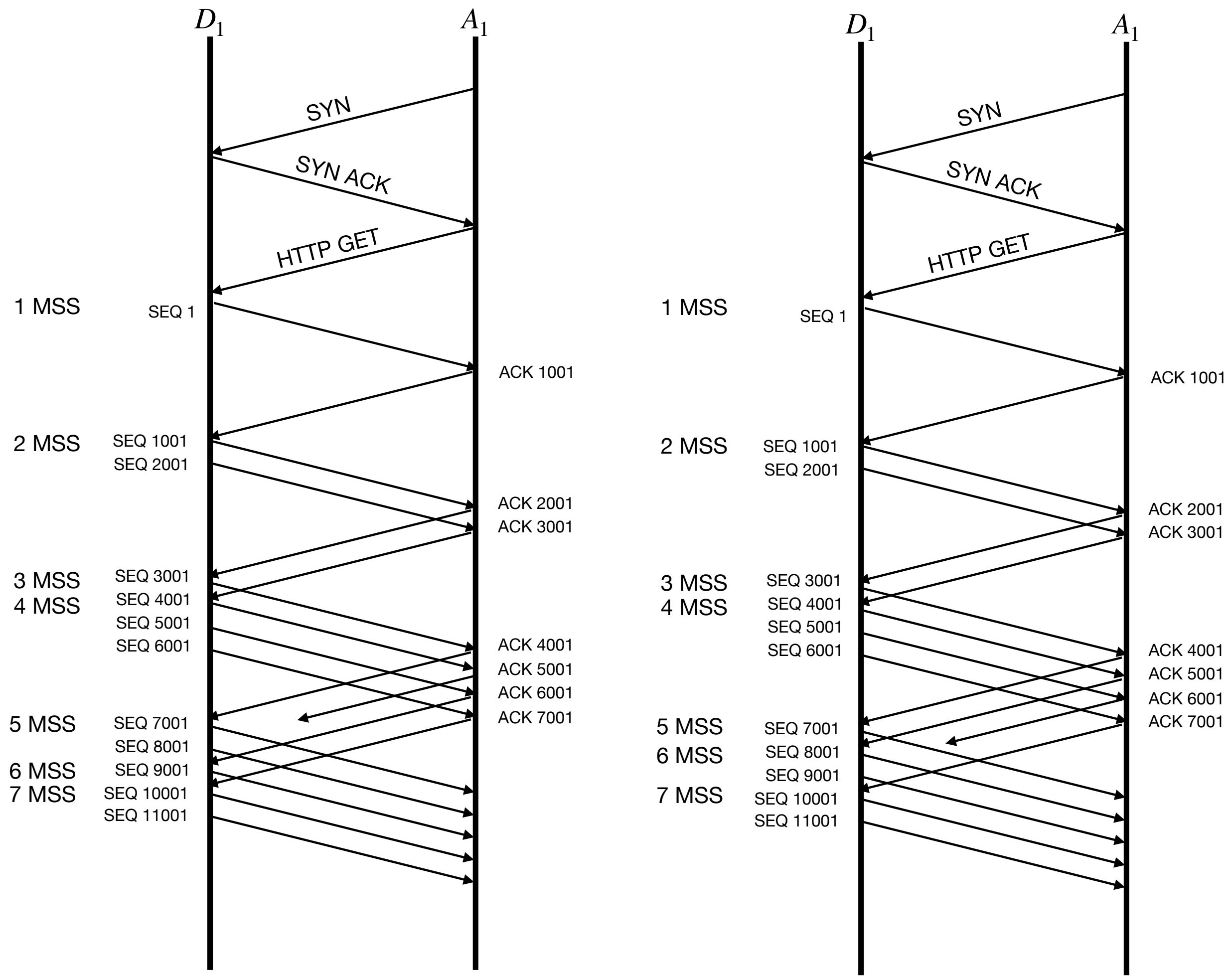


Figure 3: Diagrams for Question 5(5).

Question 3 – Forking and multi-threading [19 points]

Consider the program `forkloop.c` on the next page, compiled as follows:

```
gcc forkloop.c -o forkloop
```

In the program, `fork` and `wait` control the lifecycle of processes; `pthread_create` and `pthread_join` the lifecycles of threads, and `pthread_mutex_lock` of critical sections.

① **[4 points]** Draw a picture with the process parent-child tree for the execution of `./forkloop 3 0`. For each node in the tree, include the variables that determine the future execution of the process in the form $var = val$.

② **[1 point]** Provide one possible output of `./forkloop 3 0`.

③ **[2 points]** Is the output of `./forkloop 3 0` deterministic? **Justify completely**.

④ **[2 points]** Provide one possible output of `./forkloop 3 1`.

⑤ **[2 points]** Is the output of `./forkloop 3 1` deterministic. **Justify completely**.

⑥ **[1 point]** How many stacks are there (maximal value) for the process where $f(z = 2)$?

⑦ **[7 points]** Draw these stacks (for the process with $f(z = 2)$) with one thread in the critical section and the other thread blocked waiting on the mutex.
Stacks should be drawn from top to bottom (as on the hardware). Each call frame of the stack must be labeled with the name of the function (but the return IP address (RIP) does not need to be shown). The arguments and local variables of each call frame must be identified as follows:

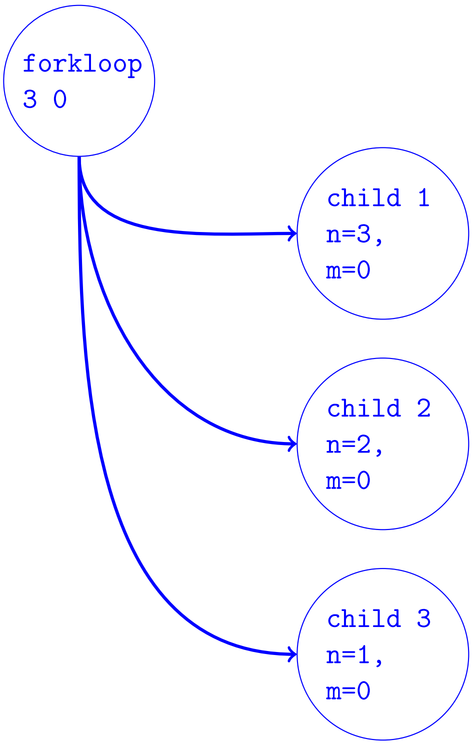
- if the variable has a known integer value you must show the name along with the value (e.g. $foo = 4$);
- if the variable points to a null-terminated string, or an array of null-terminated strings with known values, show it as $foo =$ "$bonjour$" or $foo =$ ["$hello$", "$world$"], respectively;
- if the variable is a pointer, you must draw an arrow to the pointed address on a stack, on the heap, or on the global segment;
- if the value cannot be determined in this scenario, label it with a question mark (e.g. $foo =?$).

Answers:

```c
// necessary #include <...>

#define MAX_N 12

pthread_mutex_t mut = PTHREAD_MUTEX_INITIALIZER;

void * g(void *arg) {
    int *p = (int *) arg;
    pthread_mutex_lock(&mut);
    *p = *p + 1000;
    pthread_mutex_unlock(&mut);
}

int f(int z) {
    if (z > 0 && z < MAX_N) {
        pthread_t thr[MAX_N];
        int acc = 0;
        for (int i = 0; i < z; i++) {
            pthread_create(&thr[i], NULL, g, &acc);
        }
        for (int i = 0; i < z; i++) {
            pthread_join(thr[i], NULL);
        }
        return acc;
    } else {
        return -2;
    }
}

int forkloop(int n, int m) {
    for ( ; n > 0; n--) {
        pid_t x = fork();
        if (x > 0) {
            wait(NULL);
        } else if (m > 0) {
            return f(n) + n ;
        } else {
            return n;
        }
    }
    return 0;
}

int main(int argc, char **argv)
{
    int a1 = atoi(argv[1]);
    int a2 = atoi(argv[2]);
    if (a1 < MAX_N)  {
        int y = forkloop(a1, a2);
        printf("done %d\n", y);
    }
}
```

Answers (continued):

①



②

```
done 3
done 2
done 1
done 0
```

③ It is deterministic, because the main process always needs to wait for the current child process to finish, and then starts the next for loop iteration.

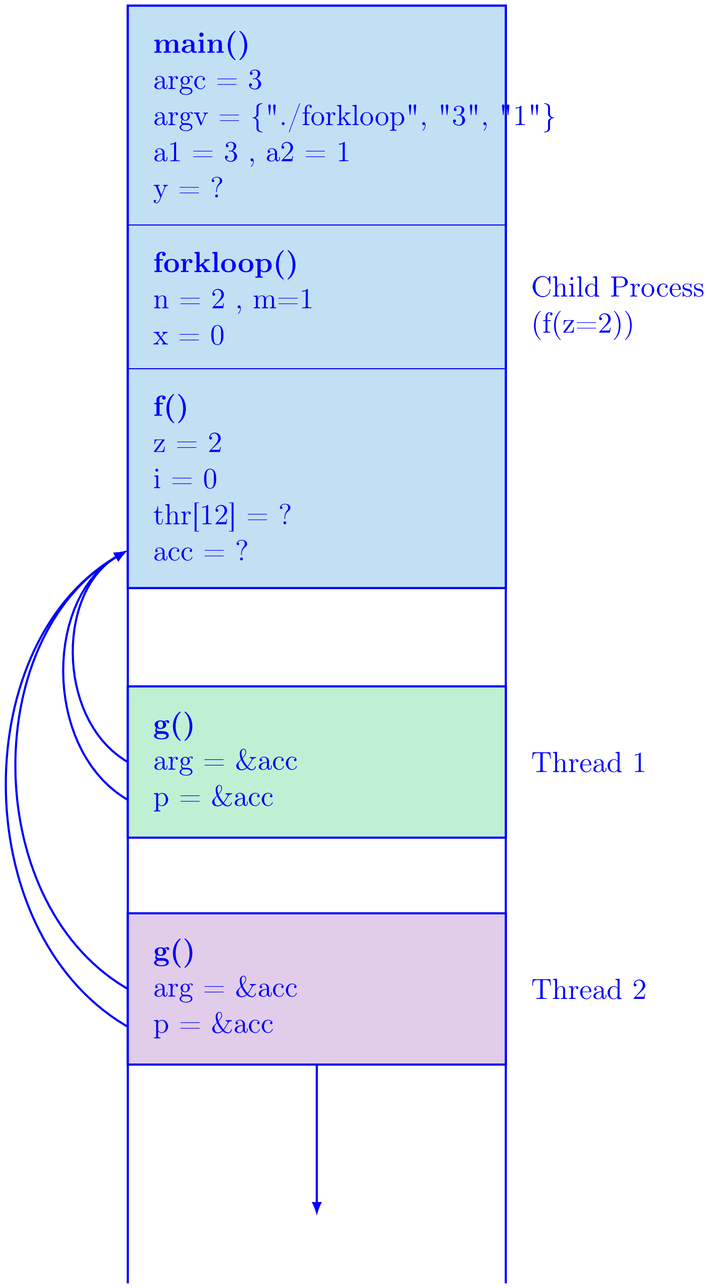
④

```
done 3003
done 2002
done 1001
done 0
```

⑤ It is deterministic. Same with ②, the main process always needs to wait for each child process to finish. Within each child process, the mutex guarantees the deterministic results of `acc`.

⑥ 3

⑦



Question 4 – File System [12 points]

Assume a file system mounted at the root directory "/" that has only the following files and directories:

```
/usr/bin/gcc
/usr/bin/clang
/bin/gcc
```

`/usr/bin/gcc` corresponds to gcc version 41 and `/bin/gcc` corresponds to gcc version 42.

Assume also that the user has full access to the whole file system. Thus, the script containing the following commands is executed without errors:

```
mv /usr/bin/clang /bin
cp /usr/bin/gcc /bin/gcc
mv /bin/clang /usr/bin
```

`mv` is the standard POSIX utility to move a file using the rename system call
(`int rename(const char *old, const char *new);`);
`cp` is the standard utility to copy a file. If the destination file already exists, `cp` opens the file with the `open` system call, and modifies its content.

Finally, assume the following hypothesis about the file system and its content:

- the inode structure has only the following:
  - a length field indicating the length of the file;
  - a modification time field;
  - entries for direct and indirect blocks;
- for simplicity, all directories and files are small and fit within a single data block;
- the OS has a very large file system buffer cache, initially empty, which is used to cache inodes and data blocks;
- the OS file system buffer cache holds recently-accessed inodes and data blocks in memory;
- the OS writes back to disk all modified inodes and data blocks synchronously on all system calls;
- the inode numbers are:

`/`: 1	`/usr`: 11	`/usr/bin`: 22	`/usr/bin/gcc`: 33
`/usr/bin/clang`: 44		`/bin`: 55	`/bin/gcc`: 66

- the next available inode numbers are: 77, 88, 99. You may or may not have to use these.

Based on the above commands (script) and assumptions:

① **[2 points]** What is the directory structure of the file system mounted at "/" once the script finishes? For `gcc`, specify the version.

Answer:

② **[10 points]** In the tables below, mark the inodes and data blocks that are read from and/or written to **disk** as a result of each command.

Enter the number of blocks associated with each inode (inode blocks, indirect blocks, data blocks) which are read from or written to **disk** for each syscall.

Mark read/write accesses to inode in table 1 and read/write accesses to data blocks in table 2 as follows:

No access: `X` One Read: `R` One Write: `W`

Examples:
One read and write: `RW` Two reads: `RR` Two reads and one write: `RRW`
and so on for different combinations of read (`R`) and writes (`W`).
The order of `RW` does not matter.

Creating/Moving a new inode/data block or updating an existing block is counted as a write.
In these tables, all blank answers will be interpreted as "*not answered*" rather than "`X`".

State your assumptions, if any.

Answers:

①

```
/usr/bin/clang
/usr/bin/gcc (v41)
/bin/gcc (v41)
```

②

inode blocks:

	In 1	In 11	In 22	In 33	In 44	In 55	In 66	In 77	In 88	In 99
`mv`	R	R	RW	X	X	RW	X	X	X	X
`cp`	X	X	X	R	X	X	RW	X	X	X
`mv`	X	X	W	X	X	W	X	X	X	X

data blocks:

	In 1	In 11	In 22	In 33	In 44	In 55	In 66	In 77	In 88	In 99
`mv`	R	R	RW	X	X	RW	X	X	X	X
`cp`	X	X	X	R	X	X	W	X	X	X
`mv`	X	X	W	X	X	W	X	X	X	X

Question 5 – Scheduling [14 points]

Consider a system with a single CPU for computation and a single disk for IO, with the following characteristics:

- CPU:
  - Round Robin (RR) policy for scheduling computation requests (i.e., with a single queue);
  - scheduling quantum: 1 second;
  - when multiple tasks enter the tail of the pending queue at the same time, they enter in the following priority (i.e., the pending queue is a FIFO):
    - new task;
    - blocked task;
    - currently running task;
  - negligible time to send disk IO request.
- Disk:
  - elevator scan scheduling policy for scheduling IO requests;
  - has five cylinders;
  - head starts at cylinder zero and moves depending on the first request;
  - seek time to move from one cylinder to next: 1 second;
  - assume that there are no rotational or transfer latencies;
  - Scheduling policy is invoked when an IO request finishes.

The following table describes *five* tasks which do some CPU and Disk IO operations. Once the disk IO operation is completed, a task executes for the remainder of its CPU total time.
Here is the description of each column:

- Task Arrival Time: the time at which the task arrives;
- CPU Total Time: total time required for the computation of the task;
- IO Request Time: the time at which the task requests for IO; for example, if task 1 starts at time 1, it will request for IO at time 2 (=1 + 1);
- IO Access: the cylinder number accessed during the IO request.

Task ID	Task Arrival Time (in seconds)	CPU Total Time (in seconds)	IO Request Time (in seconds)	IO Access (Cylinder Number)
1	1	3	1	3
2	2	2	1	4
3	3	5	1	2
4	3	3	2	1
5	3	3	1	3

Given the list of five tasks above, fill in the time "diagram" on the next page (table) to show when each task computation and IO completes.
State your assumptions whenever necessary or appropriate.

Fill in the "time diagram as follows" from "Time=2" onwards:

- Running Task as the task that executes on the CPU during the period.
- Ready tasks is the ordered list of ready tasks during the period.
- Same for blocked tasks.
- "Current Disk Cylinder" expressed as "[Task ID:Cylinder Number]", for instance: [1:3], where Task ID is the task associated with the current disk seek and Cylinder Number should reflect the position of the disk at the *end* of period.
- "Pending Disk IO requests" as a list of [Task ID:Cylinder Number], for instance: [1:3], [1:4].

You may not need to fill all the rows of the table.

Time	Running Task (Task ID)	Ready Tasks (Tasks IDs)	Blocked Tasks (Tasks IDs)	Current Disk Cylinder	Pending Disk IO requests
0	-	-	-	[-:0]	-
1	1	-	-	[-:0]	-
2	2	-	1	[1:1]	-
3	3	4, 5	1, 2	[1:2]	[2:4]
4	4	5	1, 2, 3	[1:3]	[2:4], [3:2]
5	5	1, 4	2, 3	[2:4]	[3:2]
6	1	4, 2	3, 5	[5:3]	[3:2]
7	4	2, 5, 1	3	[3:2]	-
8	2	5, 1, 3	4	[4:1]	-
9	5	1, 3, 4	-	[-:1]	-
10	1	3, 4, 5	-	[-:1]	-
11	3	4, 5	-	[-:1]	-
12	4	5, 3	-	[-:1]	
13	5	3	-	[-:1]	
14	3	-	-	[-:1]	
15	3	-	-	[-:1]	
16	3	-	-	[-:1]	
17					
18					
19					
20					

Question 6 – C Programming [25 points]

6.1 Wrong or right? [2 points]

Provided that the library `string` has been included (and there is a proper `main()` function), would the following portion of code compile?

Fully justify your answer.

```
1   #define NAME_SIZE 127
2   #define NB_VALUES 4
3
4   struct Foo {
5     char name[NAME_SIZE+1];
6     int whatever;
7     double values[NB_VALUES];
8   };
9
10  void init_phys(struct Foo* s)
11  {
12    double values[] = { 299792458, 9.80665, 6.02214076e23, 1.602176634e-19 };
13    strcpy(s->name, "Some physics");
14    s->whatever = 42;
15    s->values = values;
16  }
```

Answer and justification:

No it does not compile since array cannot be assigned (line 15).

6.2 Pointers [8 points]

On a 64-bit architecture where:

- `sizeof(short int)` is 2, such that $256 \times a + b$ is represented in memory with b first then a;
- the integer value of `(char)'A'` is 65, the one of `'B'` is 66, etc.;

what does the following code print?

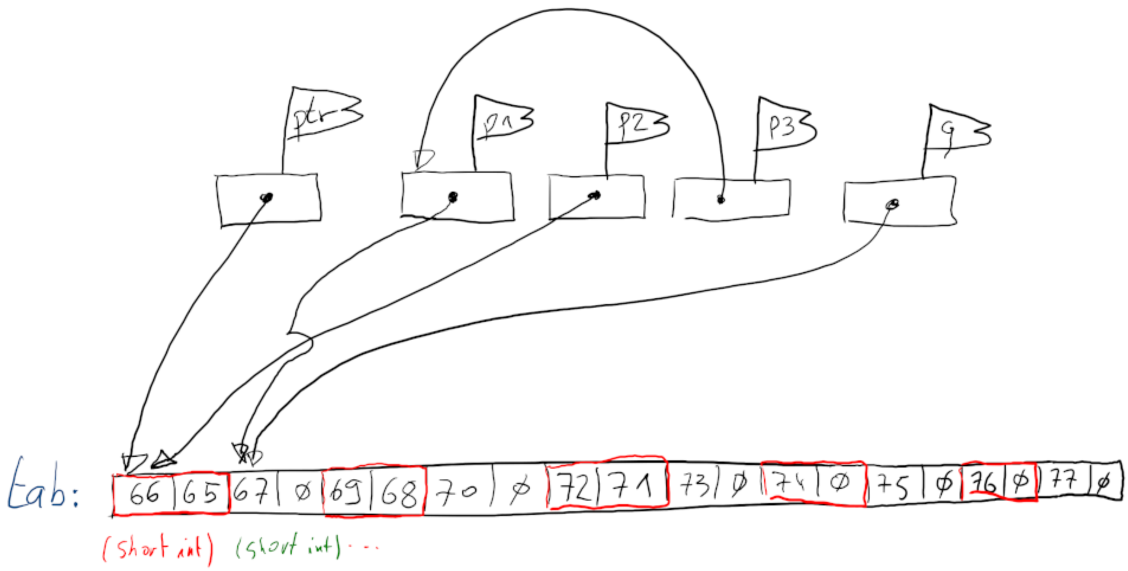
Fully justify your answer **and provide** a drawing of the memory state of the variables `tab`, `ptr`, `p1`, `p2`, `p3` and `q` at line 6 just after the call line 25.

```
1  #include <stdio.h>
2  #include <string.h>
3
4  void f(int nb, const short int* q, size_t sz)
5  {
6    printf("%d: ", nb);
7    for (size_t i = 0; i < sz; ++i) printf("%d, ", *(q + i));
8    putchar('\n');
9  }
10
11 void g(void* ptr)
12 {
13   const char* const p1 = ptr;
14   printf("1: \"%s\"\n", p1);
15
16   const short int* const p2 = ptr;
17   printf("2: %d\n", *p2);
18
19   f( 3, p2, 4           );
20   f( 4, p2, sizeof(ptr) );
21   f( 5, p2, strlen(ptr) );
22
23   const short int** p3 = &p1;
24   ++(*p3);
25   f( 6, *p3, 1);
26 }
27
28 int main(void)
29 {
30   short int tab[10] = { 65 * 256 + 66,  // 16'706
31                                   67,
32                         68 * 256 + 69,  // 17'477
33                                   70,
34                         71 * 256 + 72,  // 18'248
35                         73, 74, 75, 76, 77 };
36   g(tab);
37   return 0;
38 }
```

Answer and justification: It prints:

```
1: "BAC"
2: 16706
3: 16706, 67, 17477, 70,
4: 16706, 67, 17477, 70, 18248, 73, 74, 75,
5: 16706, 67, 17477,
6: 67,
```

Here is the corresponding memory state:



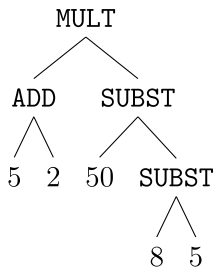
Justifications:

1. as a string (`char*`), it stops at the first null-char;
2. simply the (`short int`) value of the first (`short int`) element;
3. simply the four first (`short int`) values;
4. `sizeof(ptr) == 8` (8 bytes = 64 bits);
5. `strlen(ptr) == 3`: see point 1;
6. this one is maybe a bit more tricky: `p3` points indeed to `p1` but the `++` has the `short int*` semantics, thus move forward by one `short int`.

6.3 A bit of arithmetics [15 points]

We here consider writing *a few pieces* of a C code, the aim of which is to do arithmetic processing using (binary) tree representation.

Each node of the tree will have an operation (which can be represented as an `int`) and two operands (which are themselves (sub-)trees).
The leaves are also tree nodes which simply have a numerical value as their "operation" and two empty operands.
For instance, the arithmetic expression $(5+2)\times(50-(8-5))$ will be represented by the binary tree drawn on the right:



① **[2 points]** Assuming that the operations are for instance represented as

```
enum Operation { ADD, SUBST, MULT, DIV };
```

propose a type (data structure) to represent the arithmetic binary trees:

Here is a simple solution:

```
typedef struct node Node; // optional
typedef Node*       Tree; // optional

struct node {
  int   value;
  Node* left;
  Node* right;
};
```

There can be several alternatives, using more attributes (e.g. a `type` attribute), more types (e.g. using a `Leaf` type), more abstract (e.g. using `void*` genericity to handle `Node`/`Leaf` distinction), using `union`, ...).

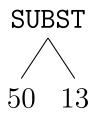
Of course, the answers to the next questions shall match with the types proposed here.

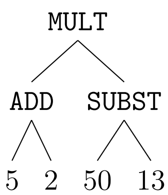
② **[1 point]** Declare a variable named `five`, stored on the stack, that would represent the leaf node 5:

```
const Node five = { 5, NULL, NULL };
```

③ **[5 points]** Define a `merge()` function that takes an operation and two trees and merge them into a higher level tree (to be returned).

For instance the merge of the tree 

with the tree 

using the operation `MULT` will return the tree: 

```
Node* merge(enum Operation op, Node* left, Node* right)
{
  Node* ret = malloc(sizeof(Node));
  if (ret != NULL) {
    ret->value = op;
    ret->left  = left;
    ret->right = right;
  }
  return ret;
}
```

④ **[1 point]** Define a `leaf()` function which takes an integer value and returns a leaf node (or a pointer to it), similar to the variable `five` from subquestion ①, but allocated on the *heap*.

`Node* leaf(int value) { return merge(value, NULL, NULL); }`

⑤ **[1 point]** Use the `merge()` and `leaf()` functions to declare a variable named `example` representing the expression $(5 + 2) \times (50 - (8 - 5))$ (the tree of which is drawn above).

```
  Node* example =
    merge(MULT,
          merge(ADD,   leaf(5) , leaf(2)),
          merge(SUBST, leaf(50),
                       merge(SUBST, leaf(8) , leaf(5))
               )
         );
```

Whatever the proposed solution (data types and `merge()`/`leaf()` implementation), I don't see how the answer to *that* question could be different.

⑥ **[5 points]** Finally, define a `release()` function that *completely* deallocates a tree, assuming all its nodes have been allocated on the heap.

```
void release(Node* tree)
{
  if (tree != NULL) {
    release(tree->left);
    release(tree->right);
    free(tree);
  }
}
```