UNIVERSITEIT VAN AMSTERDAM 2017-2018

minor programmeren

Tentamen Fulltime Programmeren 2

–

Schrijf je naam en studentnummer meteen op de regel hieronder.

____

Je mag de vragen in Engels of Nederlands beantwoorden.

Dit is een "gesloten boek"-tentamen. Je mag voor het invullen je pen of potlood gebruiken, maar verder niets. Schrijf duidelijk en niet te groot.

Leg je studentenkaart (of ander ID met foto) klaar op je tafel. We komen langs om te kijken of je hierboven je naam hebt ingevuld en of deze klopt met je ID.

Laat het weten als je kladpapier nodig hebt.

Als je vragen hebt over hoe we iets bedoelen, dan kunnen we dat waarschijnlijk niet beantwoorden zonder een deel van het antwoord weg te geven (maar voel je vrij om het te proberen!).

Je hoeft geen comments in je code te schrijven. Je mag alle aangeleverde functies uit de lectures of de opdrachten gebruiken in je antwoorden. Pseudocode kan ook punten opleveren, maar doorgaans minder dan "echte" code!

Data, structures and data structures.

For each of the questions below, circle the correct answer.

3. (1 point.) Consider an implementation of a **singly** linked list where **only** a pointer to the head is available. What is the running time for inserting an element at the tail of the list?

   a. $\Omega(1)$
   b. $\Omega(n)$
   c. $\Omega(\log n)$
   d. $\Omega(n \log n)$
   e. $\Omega(n^2)$

4. (1 point.) Say we perform the following sequence of operations on a *queue* data structure. Which numbers are *dequeued*, and in what order?

```
enqueue(1)
enqueue(2)
dequeue()
enqueue(3)
dequeue()
dequeue()
enqueue(4)
enqueue(5)
enqueue(6)
dequeue()
```

5. (3 points.) The following function is not a great hash function. Briefly give two reasons why.

```
int hash(char* word)
{
    return word[0];
}
```

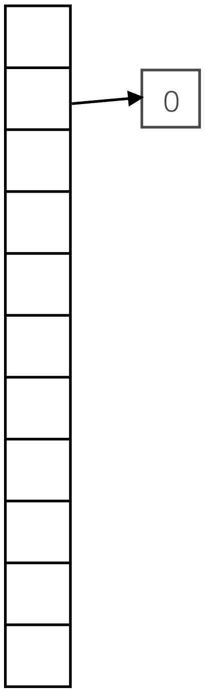
Handywork.

The following code implements a hash table and its corresponding *insert* operation.

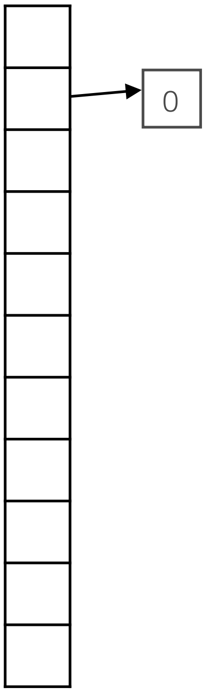
```
typedef struct node
{
     int data;
     struct node *next;
}
node;

node* hashtable[11];

unsigned int hash(unsigned int key)
{
     return (key % 4) + 1;
}

void insert(int num)
{
     node* num_node = malloc(sizeof(node));
     num_node->data = num;
     int location = hash(num);
     num_node->next = hashtable[location];
     hashtable[location] = num_node;
}
```

7. (2 points.) Consider the visual representation of our hash table's "buckets", below. We have already used the `insert` function to add the number 0 to the table. Draw the final state of the hash table after inserting these additional values: 7, 18, 3, 5, 13, 25, and 9, in that order.

(use this one to try it out)



(clearly draw your **final solution** here)



Hey, browser.

Have a look at this HTML document:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Star Lord</title>
        <link href="style.css" rel="stylesheet"/>
    </head>
    <body>
        <p class="header" id="star-lord" style="color:red">Star Lord</p>
        <p class="text">I'm pretty sure the answer is "I am Groot."</p>
        <p class="header" id="groot" style="color:orange">Groot<p>
        <p class="text">I am Groot.</p>
    </body>
</html>
```

And the accompanying CSS style sheet `style.css`:

```
.text {
    color:blue;
}
```

8. (1 point.) Which colors will be rendered for the different `p` elements? For each, circle the correct answer:

“Star Lord”	black / red / blue / orange
“I'm pretty sure the answer is "I am Groot."”	black / red / blue / orange
“Groot”	black / red / blue / orange
“I am Groot.”	black / red / blue / orange

9. (1 point.) The `p` tags for the texts “Star Lord” and “Groot” each contain a style attribute. We would like to move those rules to `style.css`, allowing us to remove the corresponding attributes from the HTML. Below, write the rules to add to `style.css`.

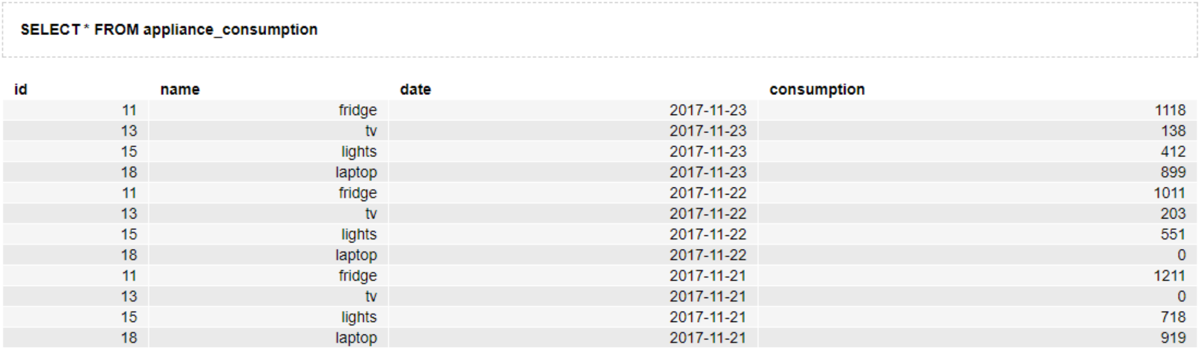
10. (1 point.) Below, write a rule that will render all the text in the body **bold**faced.

Frosh IMs.

We're monitoring the energy consumption of our household appliances. We would like to create a webpage that shows the daily energy use (in Wh) per appliance. We have already created the `/consumption` route, which queries the database and renders the corresponding template file `consumption.html`.

```
@app.route("/consumption")
def consumption():
     rows = db.execute("SELECT * FROM stats")
     total = db.execute("SELECT SUM(consumption) AS total FROM stats")
     return render_template("consumption.html", consumption=rows, total=total)
```

As an example, here is a snapshot of information that we might find in the database:



SELECT * FROM appliance_consumption

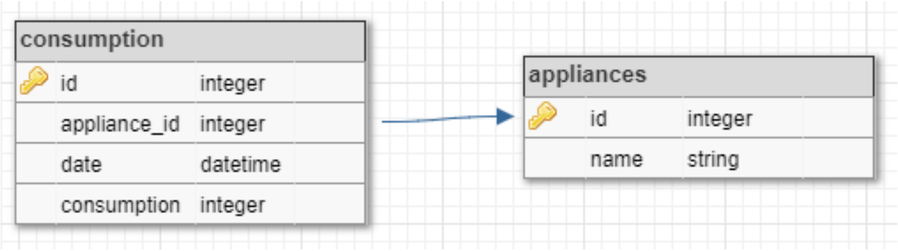
id	name	date	consumption
11	fridge	2017-11-23	1118
13	tv	2017-11-23	138
15	lights	2017-11-23	412
18	laptop	2017-11-23	899
11	fridge	2017-11-22	1011
13	tv	2017-11-22	203
15	lights	2017-11-22	551
18	laptop	2017-11-22	0
11	fridge	2017-11-21	1211
13	tv	2017-11-21	0
15	lights	2017-11-21	718
18	laptop	2017-11-21	919

11. (4 points.) Complete the implementation of consumption.html below, such that it renders a HTML table for all appliance consumption. The first column should contain the appliance name, the second column the date, and the last column the consumption. Recall that an HTML table is defined with the `<table>` tag. Each table row is defined with the `<tr>` tag. A table cell is defined with the `<td>`. The last column of the last row of the table should contain the total energy consumption of all appliances.

```
{% extends "layout.html" %}

{% block title %}
    Energy consumption
{% endblock %}

{% block body %}
<table>
```

```
</table>
{% endblock %}
```

We have improved the database design of the previous assignment. Instead of one table, we have created two tables, like so:



consumption	
id	integer
appliance_id	integer
date	datetime
consumption	integer

appliances	
id	integer
name	string

12. (2 points.) Why is this a better database design?

13. (2 point.) Write a SQL query that selects only the name of all appliances.

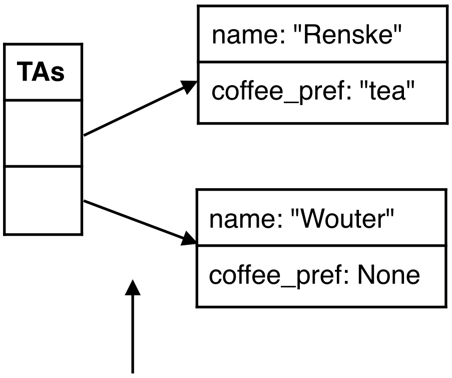
14. (1 points.) We would like to know the total consumption of every appliance. Which of the following queries would yield that information?

15. (2 points.) Write a SQL query that inserts today's consumption (532 Wh) of an appliance with `id` 15. Ignore the `date` column, and assume it will automatically be set to today's date when inserting a record.

Coffee habits.

Consider the following Python class, which is used to make sure that the coffee machine dispenses the right beverage for each of the teaching assistants.

```
class TA(object):
    def __init__(self, name, coffee_pref=None):
        self.name = name
        self.coffee_pref = coffee_pref
    def change_preference(self, beverage):
        self.coffee_pref = beverage
    def __str__(self):
        raise NotImplementedError
```



like in this diagram

16. (3 points.) Complete the following code to create two TA objects, like in the diagram to the right, and to add them to a list:

```
TAs = []
TA1 = TA(
```

Say we have added all other staff to the list, thereby completing the coffee preferences administration. Due to a bug in the system, former assistant Ben has managed to hack the database and change all of the staff's coffee preferences to "Wiener Melange". How cruel!

17. (3 points.) Write code to remedy the situation and change the preferences for all staff members in the `TA` list to "Hot Chocolate", preparing for the upcoming holidays.

Now, `__str__` is a special method, that generates the string that will output when `print`ing the object:

```
>> TA1 = TA(..)     # initialize one TA object with name Simon and preference
>> print(TA1)       # prints "Simon likes Hot Chocolate"
```

18. (2 points.) In the class definition above, `__str__` is not implemented yet. Which of the following lines do we need to insert into the `__str__` method to make sure the print function does indeed print `Simon likes Hot Chocolate` in this case?

a. `return '{} likes {}'.format(name, coffee_pref)`

b. `return '{} likes {}'.format(self.name, self.coffee_pref)`

c. `print ('{} likes {}'.format(name, coffee_pref))`

d. `print ('{} likes {}'.format(self.name, self.coffee_pref))`