Løsningsforslag for oppgavesett 1

Kodekveld den 23.01.2020

USN Kongsberg

Revisjon 1.0

Innholdsfortegnelse

1 Vanskelighetsgrad: Nivå 1

1.1 Regndråper

```python
def rainDrop(number):

    # Tom string vi fyller med Pling, Plang, Plong eller tallet
    str_print = ''

    # Faktor av 3
    if number % 3 == 0:
        str_print += 'Pling'

    # Faktor av 5
    if number % 5 == 0:
        str_print += 'Plang'

    # Faktor av 7
    if number % 7 == 0:
        str_print += 'Plong'

    # Hvis tallet ikke har noen av faktorene
    if str_print == '':
        str_print = str(number)

    return str_print


# Printer ut svaret til funksjonen
print(rainDrop(8))
```

1.2 Scrabble Score

```
1
2  def scrabbleScore(word):
3
4      score = 0
5
6      # Legger til arrays som inneholder alle bokstav-gruppene
7      alpha_collection_1 = ['A', 'E', 'I', 'O', 'U', 'L', 'N', 'R', 'S', 'T']
8      alpha_collection_2 = ['D', 'G']
9      alpha_collection_3 = ['B', 'C', 'M', 'P']
10     alpha_collection_4 = ['F', 'H', 'V', 'W', 'Y']
11     alpha_collection_5 = ['K']
12     alpha_collection_6 = ['J', 'X']
13     alpha_collection_7 = ['Q', 'Z']
14
15     # Kjører gjennom hver bokstav i ordet
16     for letter in word:
17
18         # Kjører gjennom array 1
19         for c in alpha_collection_1:
20             # Sjekker om en bokstav i ordet tilsvarer en bokstav i array 1 eller en lowercase
                   versjon av det.
21             # I ascii-tabellen er det 32 desimalverdier som skiller små fra store bokstaver
22             if letter == c or letter == chr(ord(c) + 32):
23                 score += 1
24
25         for c in alpha_collection_2:
26             if letter == c or letter == chr(ord(c) + 32):
27                 score += 2
28
29         for c in alpha_collection_3:
30             if letter == c or letter == chr(ord(c) + 32):
31                 score += 3
32
33         for c in alpha_collection_4:
34             if letter == c or letter == chr(ord(c) + 32):
35                 score += 4
36
37         for c in alpha_collection_5:
38             if letter == c or letter == chr(ord(c) + 32):
39                 score += 5
40
41         for c in alpha_collection_6:
42             if letter == c or letter == chr(ord(c) + 32):
43                 score += 8
```

```
44
45          for c in alpha_collection_7:
46              if letter == c or letter == chr(ord(c) + 40):
47                  score += 10
48
49      return score
50
51  # Printer ut svaret til funksjonen
52  print(scrabbleScore('Kodesonen'))
```

1.3 Romalder

```
1   # Omgjøring fra år til sekunder
2   def toSeconds(number):
3       return number * 365.25 * 24 * 60 * 60
4
5
6   # Omgjøring fra sekunder til år
7   def toYears(number):
8       return number / 60 / 60 / 24 / 365.25
9
10
11  def spaceAge(age):
12
13      # Alle brøkdeler sammenliknet med jordens banetid for hver planet
14      earth_time = 1
15      mercury_time = 0.2408467
16      venus_time = 0.61519726
17      mars_time = 1.8808158
18      jupiter_time = 11.862615
19      saturn_time = 29.447498
20      uranus_time = 84.016846
21      neptun_time = 164.79132
22
23      age = toYears(age)
24
25      # Ganger alderen med hver brøkdel tilsvarende hver planet
26      print('Earth: %f \n Mercury: %f \n Venus: %f \n Mars: %f \n Jupiter: %f \n Saturn: %f \n
            Uranus: %f \n Neptun: %f'
27          %(age * earth_time, age * mercury_time, age * venus_time, age * mars_time, age *
                jupiter_time, age * saturn_time, age * uranus_time, age * neptun_time))
28
29
30  # Skriv inn din alder
31  spaceAge(toSeconds(20))
```

1.4 Vektskål

```
def balancing(scale, options):

    # Sjekker om skålen allerede er balansert
    if scale[0] == scale[1]:
        return 'Already balanced'

    # Kjører gjennom hvert alternativ
    for i in range(0, len(options)):

        # Kjører gjennom hvert alternativ unntatt oppbrukte verdier fra den ytre loopen
        for j in range(i + 1, len(options)):

            # Sjekker om venstre eller høyre side kan balanseres med en verdi
            if (scale[0] + options[i] == scale[1]) or (scale[1] + options[i] == scale[0]):
                return '%d' % options[i]

            # Sjekker om venstre eller høyre side kan balanseres med to verdier
            elif (scale[0] + options[i] + options[j] == scale[1]) or (scale[1] + options[i] +
                 options[j] == scale[0]):
                return '%d,%d' % (options[i], options[j])

            # Skjekker om venstre og høyre side kan balanseres med ulike verdier
            elif (scale[0] + options[i] == scale[1] + options[j]) or (scale[1] + options[i]
                 == scale[0] + options[j]):
                return '%d,%d' % (options[i], options[j])

    # Hvis ingen verdier balanserer skålen
    return 'Not possible'


# Printer ut svaret av funksjonen
print(balancing([0, 14], [1, 2, 7, 7]))
```

2 Vanskelighetsgrad: Nivå 2

I denne oppgaven skal du programmere ditt eget hangman-spill, men oppgaven består av noen spesifikke krav. Du får ikke lov til å velge (eller hardkode) dine egne "riktige ord" (fasit-ord), men istedenfor må du bruke vår API for å hente ut et tilfeldig ord som skal brukes som "det riktige ordet" i spillet ditt.

Altså brukeren skal gjette på det riktige ordet, men dette ordet er hentet fra vår API. Du kan få ord som "kaffe", "sjokolade", "dronesonen", osv. Disse ordene genereres helt tilfeldig.

I løsningsforslaget under er oppgaven løst i C++. I C++ finnes det veldig mange forskjellige metoder for å lese innhold fra en nettside, men vi anbefaler å bruke **cURL**. Det kan hende at du må laste ned biblioteket, om det ikke allerede er installert på maskinen din. Les mer om cURL her: https://curl.haxx.se.

Når du har lastet ned innholdet fra API-en så ønsker du å behandle denne informasjonen og hente ut informasjon som du kan bruke videre i programmet. Slik det ble nevnt tidligere så anbefaler vi å bruke JSON for å behandle denne dataen. Det kan hende at du må laste ned biblioteket, om det ikke allerede er installert på maskinen din. Isåfall kan det lastes ned her: https://github.com/nlohmann/json.

Den største utfordringen med denne oppgaven er kanskje å lese innhold til et buffer, og å konvertere bufferet til et JSON objekt. Det finnes flere måter å gjøre dette på, men i dette eksempelet bruker vi noen innebygde funksjoner skrevet av cURL for å lese content.

Fremgangsmåten er kommentert i koden på neste side.

```cpp
#include <iostream>
#include <string>
#include <curl/curl.h>
#include <nlohmann/json.hpp>

static size_t WriteCallback(void *contents, size_t size, size_t nmemb, void *userp)
{
    ((std::string*)userp)->append((char*)contents, size * nmemb);
    return size * nmemb;
}

void setAllZeros(char arr[], int size)
{
    for(int i = 0; i < size; i++) {
        arr[i] = '_';
    }
}

void hangman(std::string secret, int count)
{
    std::cout << "Ditt ord bestr av " << count << " bokstaver." << std::endl;
    int points = 0;

    char guessed[count];
    setAllZeros(guessed, count);

    while(true) {
        std::cout << "Gjett en bokstav: ";
        char c; std::cin >> c;

        for(int i = 0; i < count; i++) {
            if(secret[i] == c){
                guessed[i] = c;
                points++;
            }
        }

        std::cout << guessed << std::endl;
        if (points == count) {
            std::cout << "Du vant!\n";
            break;
        }
    }
}
```

```cpp
int main(void)
{
    CURL *curl;
    CURLcode res;
    std::string readBuffer;
    std::string secretWord;
    int wordCount;

    curl = curl_easy_init();

    if (curl) {
        // Lese content inn til readBuffer
        curl_easy_setopt(curl, CURLOPT_URL, "https://api.kodesonen.no/?task=hangman");
        curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, WriteCallback);
        curl_easy_setopt(curl, CURLOPT_WRITEDATA, &readBuffer);
        res = curl_easy_perform(curl);
        curl_easy_cleanup(curl);

        // Gjør om string readBuffer til json objekt
        nlohmann::json json_object = nlohmann::json::parse(readBuffer);

        // Parse ut alle strings
        auto json_parse = json_object.get<std::unordered_map<std::string, nlohmann::json>>();

        // Sjekk om left-side string er random-word.  Isåfall hent ut right-side.
        for (auto i : json_parse) {
            if(i.first == "random-word") secretWord = i.second;
            if(i.first == "characters") wordCount = i.second;
        }

        hangman(secretWord, wordCount);
    }
    return 0;
}
```

3 Vanskelighetsgrad: Nivå 3

Den 23. januar ble det publisert et bilde på Facebook med en kryptisk tekst. Teksten inneholdt følgende:

RGV0IGZpbm5lcyAzIHNranVsdGUgcHJpbXRhbGwgaSBkZXR0ZSBiaWxkZXQsIG11bHRpcGxpc2
VyIGRpc3NlIHNhbW1lbiBvZyBicnVrIGRlIHNvbSBlbiBVUkwgZm9yIEtvZGVzb25lbi4=

Dette er Base64, en såkalt "binary-to-text encoding", som dekodes til å bli:

"Det finnes 3 skjulte primtall i dette bildet, multipliser disse sammen og bruk de som en URL for Kodesonen."

De to første primtallene fant en ved å se på dimensjonene til bildet, det var 503 piksler i bredden og 509 piksler i høyden. Det siste primtallet 65537 var skjult i selve bildet. For å finne dette måtte en ta med bildet inn i et redigeringsprogram og justere på farge- og lysinnstillingene.

Ved å multiplisere sammen primtallene fikk en tallet 16779241499. Med dette tallet trengte en å legge til en URL for kodesonen: https://kodesonen.no?side=16779241499. På denne siden dukket det en ny tekst opp:

"Nwqm qwoulb. Axzbr lamt opt itlzr th ont aqy isixtama ihi pttxhl Rwazzjrkn. Lrm lz ahl abf lz fdqcym iix wlv."

På kildekoden til siden er det en HTML kommentar "CSS variabelen har et rart navn", denne ble navngitt "blaise_de". Ved et kjapt Google søk på navnet dukker det opp en person, nemlig Blaise de Vigenère. Utifra det blir det hintet om at en skal bruke Vigenere dekryptering på teksten som har blitt oppgitt.

Ved å gå tilbake til det originale bildet finner en ut at kodeordet for Vigenere dekrypteringen er skjult ved et akrostikon av de første bokstavene i linjene på bildet, disse blir lagt sammen til "HINT". Utifra denne informasjonen får en dekryptert chifferet til å bli:

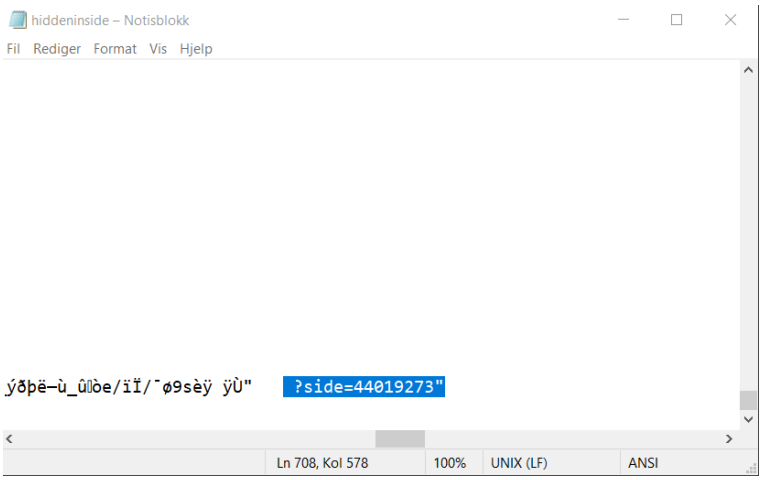
"Godt jobbet. Neste steg vil vaere aa gaa til plakaten paa campus Kongsberg. Det er noe som er skjult bak den."

Bak Kodesonen plakaten i Dronesonen på campus Kongsberg hang det en A4 side med en QR kode:

QR koden ville ta en videre til en ny side: https://kodesonen.no?side=299414291677. Her ble en møtt med en form som spurte om et passord for å komme seg videre:

For å finne passordet måtte en se på kildekoden igjen. Her fant en ut at passordet var hardkodet inn på "client-side" og var simpelthen "Javascript". Ved å taste inn passordet og klikke på "Login" ble en tatt med videre til en ny side: https://kodesonen.no?side=30366452.

På denne siden fikk en opp teksten "Enkelte ganger burde man se forbi det visuelle. Bilder kan inneholde klartekst." og et bilde. Ved å åpne bildet i et tekstredigeringsverktøy eller bruke en "string extraction" på bildet fikk en opp enda en ny URL: https://kodesonen.no?side=44019273.



Denne siden var den aller siste. Her ble en møtt med et C++ program som ikke kompilerte og teksten "Greider du å finne feilen med logikken i kildekoden nedenfor? OBS! Kan være du får et lite anagram. Når du har løst det kan du sende en mail til kontakt@kodesonen.no".

Logikken til programmet er løst under løsningsforslag for nivå 3 "oppgave3_loesning.cpp" på Github. Ved å få det til å kompilere ville en fått anagrammet "CLRT" som blir om til "CTRL". En annen godkjent måte å løse oppgave 3 var å ikke nødvendigvis få programmet til å kompilere, men å derimot se på hva det kunne ha blitt kompilert til. Dette kunne en gjøre ved å se primtallene 2, 11, 17 og 19 og få det til å passe sammen med det vedlagte fonetiske alfabetet til Nato.

Vinneren ble oppgitt på Facebook og vant en Kodesonen T-skjorte.