# Practicum woensdag: dynamisch geheugen



Voor de automatische verbetering op INGinious is het belangrijk dat je na elke cout ook een endl toevoegt!

Your answer passed the tests! Your score is 100.0%. [Submission #673f4cb3de404ce80b3af62f (**2024-11-21 15:07:31**)]

## Question 1: Memory leaks

✓ Perfect

1. Het volgende programma heeft een memory leak. Als je het runt (klik op submit), krijg je een rapport van valgrind. Als je goed kijkt, zie je dat er effectief geheugen niet wordt vrijgegeven. Dat duidt op een memory leak.
2. Een typische fout is het toevoegen van `delete a`. Doe dit en submit opnieuw. Je zal merken dat er nu minder geheugen verloren is, maar er is nog steeds een memory leak.
3. Los nu de memory leak volledig op.

```
1  int main(){
2      int* a = new int[10];
3      delete[] a;
4      return 0;
5  }
```

Submit

## Question 2: Je eigen garbage collector (a)

✓ Perfect

In C++ is het de verantwoordelijkheid van de programmeur om zelf het dynamische geheugen vrij te geven. In Python gebeurde dat achter de schermen voor jou. We kijken nu hoe we dat zelf kunnen implementeren. We doen dat door de gebruiker geen new en delete te laten gebruiken, maar we schrijven onze eigen functies die dat doen. We gaan er even vanuit dat het dynamisch aangevraagd geheugen enkel arrays van integers zijn. Er is reeds een globale variabele `vector<int*> memory` aangemaakt die het dynamisch geheugen bijhoudt. Die is initieel leeg. Een gebruiker kan nieuw geheugen aanvragen door een functie `allocate` aan te roepen (vergelijkbaar met de new), maar hij moet geen delete doen na elke allocate. Op het einde van het programma kan hij de functie `free` oproepen die al het dynamisch geheugen vrijgeeft zonder dat er memory leaks zijn.

Schrijf nu een functie `allocate(n)` die een array van `n` integers aanmaakt en teruggeeft. Bovendien registreert die dat er nu dynamisch geheugen gealloceerd is. Het registreren gebeurt door de array toe te voegen aan de variabele `memory`.

```
1  int* allocate(int n)
2  {
3      int* result = new int[n];
4      memory.push_back(result);
5      return result;
6  }
```

Submit

## Question 3: Je eigen garbage collector (b)

✓ Perfect

Schrijf een functie `free()` die al het dynamisch geheugen dat gealloceerd is, vrij geeft. Valgrind wordt opnieuw gerund om te testen op memory leaks.

```
int free()
{
    for (auto& i : memory){
        delete[] i;
    }
    memory.clear();
}
```

Submit

## Question 4: Statistische analyse (a)

✓ Perfect

We schrijven een programma voor statistische analyse zonder gebruik te maken van een vector. We gebruiken arrays.

Schrijf een functie `gemiddelde(a,n)` die het gemiddelde berekent van een array van integers `a` en lengte `n`. Denk eraan dat je een array doorgeeft als een pointer naar het eerste element. Let ook op voor conversies. Als je in C++ een int deelt door een int krijg je ... een int! Je lost dat op door een van de operands om te zetten in een double (bv * 5.0 i.p.v. * 5) of door expliciet te casten: (double) 5 = 5.0.

```
double gemiddelde(int* a, int n)
{
    double result = 0;
    for (int i = 0; i < n; ++i)
        result += *(a+i);

    return result / (double) n;
}
```

Submit

## Question 5: Statistische analyse (b)

✓ Perfect

Schrijf een functie `mediaan(a,n)` die de mediaan berekent van een *gesorteerde* array van integers `a` en lengte `n`. Denk eraan dat je een array doorgeeft als een pointer naar het eerste element.

```
double mediaan(int* a, int n)
{
    return n % 2 ? *(a+(n/2)) : (*(a+n/2) + *(a+n/2-1))/2.0;
}
```

Submit

## Question 6: Statistische analyse (c)

✓ Perfect

Schrijf een functie `leesMetLengte` en dient om een array van getallen aan te maken en in te lezen. De functie geeft de array EN de lengte van de array terug. Merk op dat je in C++ je maar 1 waarde kan teruggeven. We geven bijgevolg niets terug en gebruiken ... Denk eraan dat je de array dynamisch zal moeten aanmaken.

In de functie roep je de reeds bestaande functie `leesGetal` op die een andere integer zal geven telkens je die oproept. De eerste integer is de lengte van de array, de volgende zijn de waarden van de array zelf in oplopende volgorde (dus ze zijn al gesorteerd). De functie werkt als volgt:

```
int main(){
    int aantal = leesGetal(); //geeft bv 3
    cout << "We lezen " << aantal << " getallen in die reeds gesorteerd zijn: " << leesGetal() << leesGetal() << leesGetal() <<
endl;
}
```

```
1  void leesMetLengte(int*& a, int& n){
2      n = leesGetal();
3      a = new int[n];
4      for(int i = 0; i < n; i++)
5          *(a+i) = leesGetal();
6  }
```

Submit

## Question 7: Statistische analyse (d)

✓ Perfect

Schrijf nu een main functie waarin je de code van de vorige vraag gebruikt om de mediaan en het gemiddelde af te drukken als volgt.

```
Gemiddelde: 10
Mediaan: 14
```

De code van de vorige vragen wordt automatisch geïncludeerd. Denk eraan dat je geen memory leaks mag hebben. Valgrind wordt opnieuw gerund om te testen op memory leaks.

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main()
6  {
7      int* a;
8      int n;
9      leesMetLengte(a, n);
10
11
12     cout << "Gemiddelde: " << gemiddelde(a, n) << endl;
13     cout << "Mediaan: " << mediaan(a, n);
14
15     delete[] a;
```

```
16 }
```

Submit

## Question 8: Statistische analyse (e)

✓ Perfect

Het is niet handig om telkens een array EN de lengte ervan door te geven. Maak een struct `MyArray` die zowel de array als de lengte bevat. Herschrijf nu de functies `gemiddelde`, `mediaan` en `leesMetLengte` en gebruik de main zoals in de vorige vraag. De functies gaan nu de struct gebruiken om door te geven (als parameter of return type). Merk op dat de `leesMetLengte` nu wel kan werken met een return! In CLion moet je de oude code niet verwijderen. In C++ mag je de naam van een functie hergebruiken zolang de parameters of het return type anders zijn. Valgrind wordt opnieuw gerund om te testen op memory leaks.

```
1  #include <iostream>
2
3  using namespace std;
4
5  struct MyArray{
6      int* array;
7      int n;
8  };
9
10 double gemiddelde(MyArray a) {
11     double result = 0;
12     for (int i = 0; i < a.n; ++i)
13         result += *(a.array+i);
14
15     return result / (double) a.n;
16 }
17
18 double mediaan(MyArray a)
19 {
20     return a.n % 2 ? *(a.array+(a.n/2)) : (*(a.array+a.n/2) + *(a.array+a.n/2-1))/2.0;
21 }
22
23 void leesMetLengte(MyArray& a){
24     a.n = leesGetal();
25     a.array = new int[a.n];
26     for(int i = 0; i < a.n; i++)
27         *(a.array+i) = leesGetal();
28 }
29
30 int main()
31 {
32     MyArray a;
33     leesMetLengte(a);
34
35
36     cout << "Gemiddelde: " << gemiddelde(a) << endl;
37     cout << "Mediaan: " << mediaan(a);
38
39     delete[] a.array;
40 }
```

Submit

## Question 9: Verwijder het maximum

✓ Perfect

Schrijf een functie `verwijderMax(a,n)` die een array `a` krijgt met lengte `n`. Je mag de ingebouwde functies van C++ niet gebruiken. De array bevat enkel positieve gehele getallen. De functie geeft een nieuwe array terug met het maximum verwijderd. Denk eraan dat je een nieuwe array moet maken met 1 item minder. Valgrind wordt opnieuw gerund om te testen op memory leaks.

```cpp
#include <iostream>

using namespace std;

// hier komt jouw code
int* verwijderMax(int* a, int n)
{
    int result[n-1];
    int max = a[0];
    int maxi = 0;
    for (int i = 0; i < n; ++i)
        if (a[i] > max) {max = a[i]; maxi = i;}

    for (int j = maxi; j < n; ++j) {
        a[j] = a[j+1];
    }
    return a;
}

// vertrek van deze main
int main(){
    int a[] = {1,5,8,2};
    int* b = verwijderMax(a,4);
    for(int i = 0; i < 3; i++){
        cout << b[i] << endl;
    }

    return 0;
}
```

Submit