Practicum vrijdag: pointers en files

Collapse context

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 #67376cf66c698110d4c20166 (**2024-11-15 15:47:02**)]

Question 1: Debuggen

✓ Perfect

Er is iets mis met volgende code. Los het op door enkel de functie `absvalue` aan te passen.

```
#include <iostream>
 using namespace std;

 // bereken de absolute waarde van a
 void absvalue(int a){
    if(a < 0){
        a = -a;
    }
 }

 void abs(int a[],int n){
     for(int i = 0; i < n; i++){
         absvalue(a[i]);
     }
 }

 void print(int a[],int n){
     for(int i = 0; i < n; i++){
         cout << a[i] << endl;
     }
 }

 int main(){
     int a[]  = {1,-3,-6,2};
     abs(a,4);
     print(a,4);
     return 0;
 }
```

```
1  void absvalue(int& a){
2      if(a < 0){
3          a = -a;
4      }
5  }
```

Submit

Question 2: Vierkantsvergelijkingen (a)

✓ Perfect

Je gaat nu een aantal functies schrijven om vierkantsvergelijkingen op te lossen. Een vierkantsvergelijking is van de vorm $ax^2 + bx + c$.

Schrijf eerst een functie `computeDiscriminant` die drie integers (a, b en c) via call by value meekrijgt om de discriminant D (een int) te berekenen $D = b^2 - 4ac$.

```
#include <iostream>
#include <cmath>
using namespace std;

int computeDiscriminant(int a, int b, int c)
{
    return b*b - 4 * a * c;
}
```

Submit

Question 3: Vierkantsvergelijkingen (b)

✓ Perfect

Schrijf nu een functie `solveQuadraticEquation` die beide oplossingen x1 en x2 vind voor $ax^2 + bx + c = 0$. Als input krijgt deze functie de vector met daarin de getallen a, b en c als call by reference. Als return waarde geeft de functie het aantal oplossingen terug. De functie uit de vorige vraag wordt automatisch toegevoegd aan je code. Als D < 0 is het aantal oplossingen 0. Als D=0 is er 1 oplossing voor de vergelijking met x1=x2. Als D>0 dan zijn er twee oplossingen voor de vergelijking. Voeg extra parameters toe (door een call by reference) aan de functie om de oplossingen x1 en x2 op te slaan (allebei met type double). (De oplossingen zijn gegeven door $\frac{-b \pm \sqrt{D}}{2a}$.)

```
#include <iostream>
#include <cmath>
#include <vector>
using namespace std;

int solveQuadraticEquation(vector<int>& g, double& x1, double& x2)
{
    int d = computeDiscriminant(g[0], g[1], g[2]);
    if (d == 0)
    {
        x1 = -g[1] / (2 * g[0]);
        return 1;
    }
    else if (d > 0)
    {
        x1 = (-g[1] + sqrt(d)) / (2 * g[0]);
        x2 = (-g[1] - sqrt(d)) / (2 * g[0]);
        return 2;
    }
    else
        return 0;
}
```

Submit

Question 4: Vierkantsvergelijkingen (c) - Pointers

✓ Perfect

Maak opnieuw een functie `solveQuadraticEquation` maar gebruik deze keer in plaats van een vector (met daarin de getallen voor a, b en c) een array waarin deze getallen opgeslagen staan. Denk eraan dat je een array doorgeeft door een pointer naar het eerste element.

Geef deze keer de parameters niet by reference aan de functie mee, maar gebruik in plaats daarvan pointer variables!

```
#include <iostream>
#include <cmath>
using namespace std;

int solveQuadraticEquation(int* g, double* x1, double* x2)
{
    int d = computeDiscriminant(*g, *(g+1), *(g+2));
    if (d == 0)
    {
        *x1 = -(*(g+1)) / (2 * (*g));
        return 1;
    }
    else if (d > 0)
    {
        *x1 = (-(*(g+1)) + sqrt(d)) / (2 * (*g));
        *x2 = (-(*(g+1)) - sqrt(d)) / (2 * (*g));
        return 2;
    }
    else
        return 0;
}
```

Submit

Question 5: Files (a)

✓ Perfect

Werken met files in C++ is heel eenvoudig. Dat gebeurt via streams. Je kent al de input en outputstreams om van het scherm te lezen of om naar het scherm te schrijven: cin en cout. Files werken op dezelfde manier. Alleen moet je nu de stream zelf een naam geven. Schrijven naar file doe je met een ofstream (een output-filestream) zo:

```
#include<fstream>

int main(){
    ofstream output{"test.txt"};
    output << "hallo" << endl;
}
```

Let wel op dat je bestand terechtkomt op dezelfde plek als je executable (in de map cmake-build-debug dus). Wil je dat niet, dan kan je het pad altijd aanpassen.

```
#include<fstream>

int main(){
    ofstream output{"../test.txt"};
    output << "hallo" << endl;
}
```

Lezen uit een file kan je op twee manieren doen met een ifstream (een input-filestream): per token of per regel. Een token is een opeenvolging van karakters zonder whitespace (spatie, tab, newline, ...). Als je bestand er zo uitziet:

```
a b       c
d
```

dan kan je dat uitlezen per token als volgt:

```
#include<fstream>

int main(){
    ifstream input{"test.txt"};
    string token = "";
    while(input >> token){
        cout << token << endl;
    }
}
```

De output is dus

```
a
b
c
d
```

Soms wil je regel per regel inlezen. Dat gebeurt met de functie (let op dit is geen methode!) als volgt:

```
#include<fstream>

int main(){
    ifstream input{"test.txt"};
    string line = "";
    while(getline(input,line)){
        cout << line << endl;
    }
}
```

De output is dan

```
a b       c
d
```

Op Blackboard vind je een filmpje van Prof. Calders over werken met files.

Schrijf een functie `readNumbers(filename)` die een bestandsnaam (een string) meekrijgt als parameter. De functie leest alle getallen uit het bestand en slaagt die op in een vector van integers. De functie geeft de vector terug. Denk eraan dat je bestand in de map cmake-build-debug moet staan. Het bestand ziet er als volgt uit:

```
1
3
2
6
9
```

```
#include <fstream>
#include<iostream>
#include <vector>

using namespace std;
vector<int> readNumbers(string filename)
{
    vector<int> result;
    ifstream input{filename};
    string line = "";
    while(getline(input,line)){
        result.push_back(stoi(line));
    }
    return result;
}
```

Submit

Question 6: Files (b)

✓ Perfect

Schrijf een functie `writeNumbersInReverse(filename,numbers)` die een bestandsnaam (een string) en een vector van integers meekrijgt als parameter. De functie schrijft de getallen naar het bestand maar in omgekeerde volgorde. De functie geeft niets terug. Het bestand ziet er als volgt uit:

```
9
6
2
3
1
```

Zorg dat je de vorige vraag eerst correct gemaakt hebt, want die wordt hier opnieuw gebruikt.

```
#include <iostream>
#include <vector>
#include <fstream>
#include <algorithm>


using namespace std;
void writeNumbersInReverse(string filename, vector<int>& numbers)
{
    ofstream file;
    file.open(filename);
    sort(numbers.rbegin(), numbers.rend());

    for (int n : numbers)
    {
        file << n << endl;
    }
    file.close();
}
```

Submit

Question 7: Meerdere containers (a)

✓ Perfect

In sommige situaties is het belangrijk om je data op verschillende manieren te kunnen bekijken. Het is daarbij belangrijk dat de data zelf maar 1 keer in het geheugen zit, maar de structuur van de data zit er dan verschillende keren in. A.h.v. pointers is dat perfect te doen.

Stel dat we een array hebben van integers. We willen de originele volgorde bewaren en passen die array dus nooit aan. Maar we willen ook een kopie bijhouden, een omgekeerde versie en een gesorteerde versie ZONDER de data te kopiëren. We gebruiken pointers om te verwijzen naar de oorspronkelijke data. We doen dat in verschillende stappen.

Schrijf een functie `shallowcopy` met drie parameters: resp. de originele array met de integers (de originele data), een (lege) array van pointers voor het resultaat en de grootte van de arrays. In het resultaat zitten na de functie-aanroep allemaal pointers die verwijzen naar de elementen van de originele array, in dezelfde volgorde.

```
int a[]  = {1,-3,-6,2};

// shallow copy
int* b[]  = {nullptr, nullptr, nullptr, nullptr};
shallowcopy(a,b,4);
cout << *b[0] << endl; // geeft 1, het eerste element van de originele array

// als je de originele aanpast, verandert de kopie ook (= shallow copy)
a[0] = 5;
cout << *b[0] << endl; // geeft dus nu 5
```

```
using namespace std;

void shallowcopy(int a[], int* b[], int size)
{
    for (int i = 0; i < size; i++)
        b[i] = &a[i];
}
```

Submit

Question 8: Meerdere containers (b)

✓ Perfect

Schrijf een functie `getPointer` die een array van pointers naar de originele data krijgt (het resultaat van shallowcopy bv), zijn size en een index. De functie geeft een pointer terug naar het element op index i (wat dus ook een pointer is!). Indien de index out of range is, geef je de nullpointer terug.

```
int a[]  = {1,-3,-6,2};

// shallow copy
int* b[]  = {nullptr, nullptr, nullptr, nullptr};
shallowcopy(a,b,4);
cout << **getPointer(b,4,0) << endl; // geeft 1
```

```
int** getPointer (int* a[], int size, int index)
{
    return index >= size ? nullptr : &a[index];
}
```

Submit

Question 9: Meerdere containers (c)

✓ Perfect

Schrijf een functie `print` die een array van pointers naar de originele data krijgt (het resultaat van shallowcopy bv), een begin- en een eindpointer. De functie drukt de waarden van de originele array (via de pointers) van een begin- tot en met een eind-pointer af. De waarden worden gescheiden door een spatie en je eindigt met een newline (endl).

```
int a[] = {1,-3,-6,2};
int* b[]  = {nullptr, nullptr, nullptr, nullptr};
shallowcopy(a,b,4);

print(b, getPointer(b,4,0), getPointer(b,4,3)); // toont 1 -3 -6 2
print(b, getPointer(b,4,1), getPointer(b,4,3)); // toont -3 -6 2
```

```
1  void print(int* arr[], int** a, int** b)
2  {
3      for (auto i = a; i <= b; ++i)
4          if (i != b)
5              cout << **i << ' ';
6          else
7              cout << **i ;
8      cout << endl;
9  }
```

Submit

Question 10: Meerdere containers (d)

✓ Perfect

Schrijf een functie `reversecopy` met drie parameters: resp. de originele array met de integers (de originele data), een (lege) array van pointers voor het resultaat en de grootte van de arrays. In het resultaat zitten na de functie-aanroep allemaal pointers die verwijzen naar de elementen van de originele array, in omgekeerde volgorde.

```
int a[] = {1,-3,-6,2};
int* b[]  = {nullptr, nullptr, nullptr, nullptr};
reversecopy(a,b,4);

print(b, getPointer(b,4,0), getPointer(b,4,3)); // toont 2 -6 -3 1
```

```
1  void reversecopy(int a[], int* b[], int size)
2  {
3      for (int i = 0; i < size; i++)
4          b[i] = a + size - 1 - i;
5  }
```

Submit

Question 11: Meerdere containers (e)

✓ Perfect

Schrijf een functie `findMin` die een array van pointers naar de originele data krijgt (het resultaat van shallowcopy bv), een begin- en een eindindex (geen pointers dus). De functie geeft een index terug waar de pointer zich bevindt die verwijst naar het kleinste element (van de originele data). De eindindex is een voorbij het einde.

```
int a[] = {1,-3,-6,2};
int* b[]  = {nullptr, nullptr, nullptr, nullptr};
shallowcopy(a,b,4);

int minIndex = findMin(b, 0, 4);
cout << "min zit op index " << minIndex << " = " << *b[minIndex] << endl;
// geeft: min zit op index 2 = -6
```

```
int findMin(int* arr[], int b, int e)
{
    int kleinst = 0;
    for (int i = b; i < e; i++)
        if (*arr[i] < *arr[kleinst])
            kleinst = i;

    return kleinst;
}
```

Submit

Question 12: Meerdere containers (f)

✓ Perfect

Schrijf een functie `sortedcopy` met drie parameters: resp. de originele array met de integers (de originele data), een (lege) array van pointers voor het resultaat en de grootte van de arrays. In het resultaat zitten na de functie-aanroep allemaal pointers die verwijzen naar de elementen van de originele array, in oplopende volgorde. Je gebruikt het volgende sorteeralgoritme:

1. maak een shallow copy: b
2. zoek het kleinste element in b[0] t.e.m. b[n-1]
3. verwissel het met b[0]
4. zoek het kleinste element in b[1] t.e.m. b[n-1]
5. verwissel het met b[1]
6. zoek het kleinste element in b[2] t.e.m. b[n-1]
7. verwissel het met b[2]
8. ...

```
int a[] = {1,-3,-6,2};
int* b[]  = {nullptr, nullptr, nullptr, nullptr};
sortedcopy(a,b,4);

print(b, getPointer(b,4,0), getPointer(b,4,3)); // toont -6 -3 1 2
```

```
void sortedcopy(int original_arr[], int* b[], int size)
{
    shallowcopy(original_arr, b, size);
    int i = 0;
    while (i < size)
    {
        int index = findMin(b, i, size);
        auto temp = b[i];
        b[i] = b[index+i];
        b[index+i] = temp;
        i++;
    }
}

```

Submit