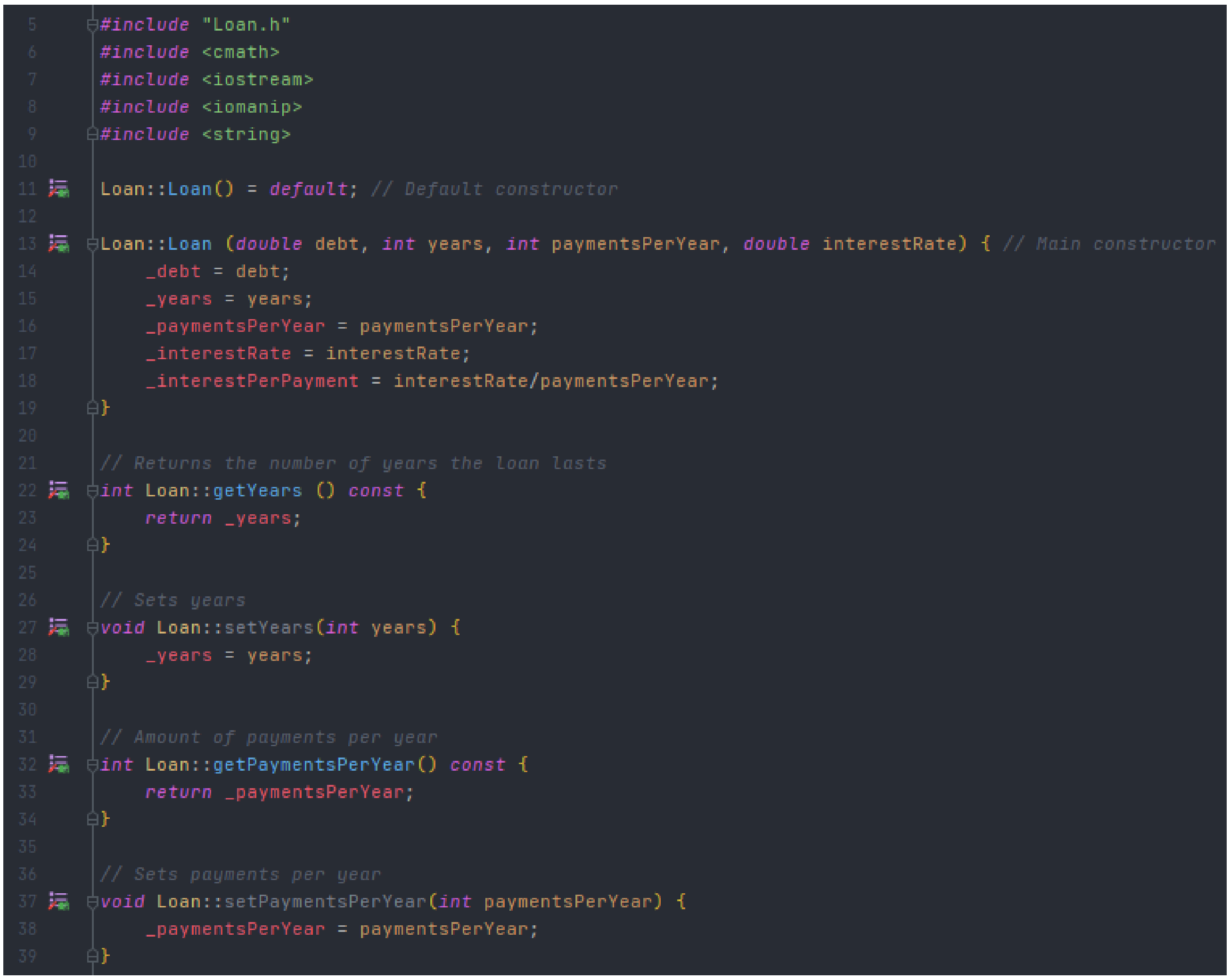
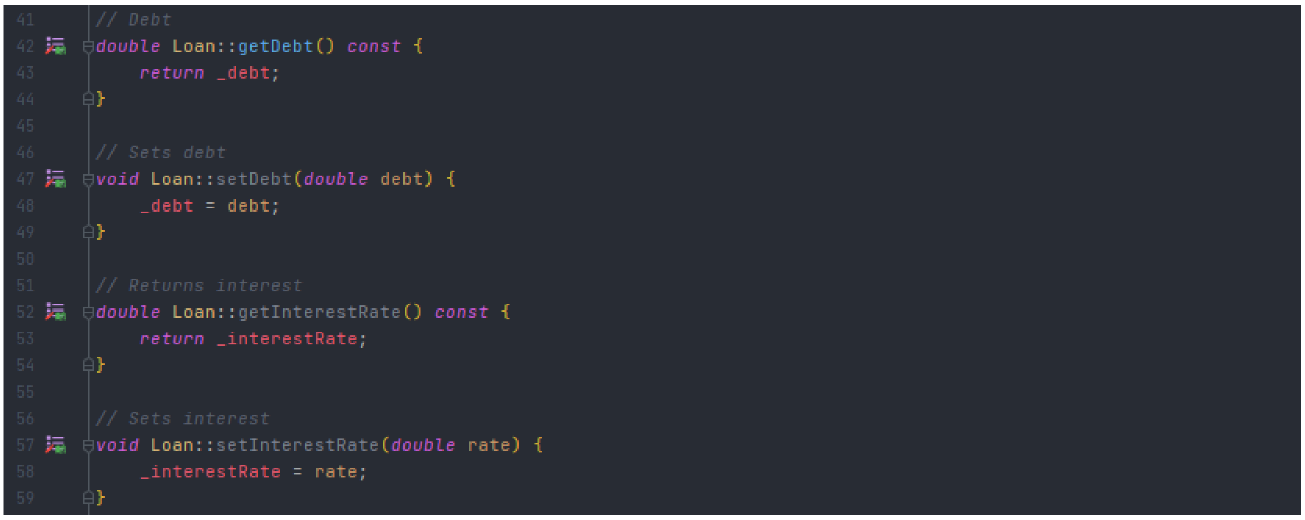
Dette er et dokument med løsningen på opgave 9.1 i C++.

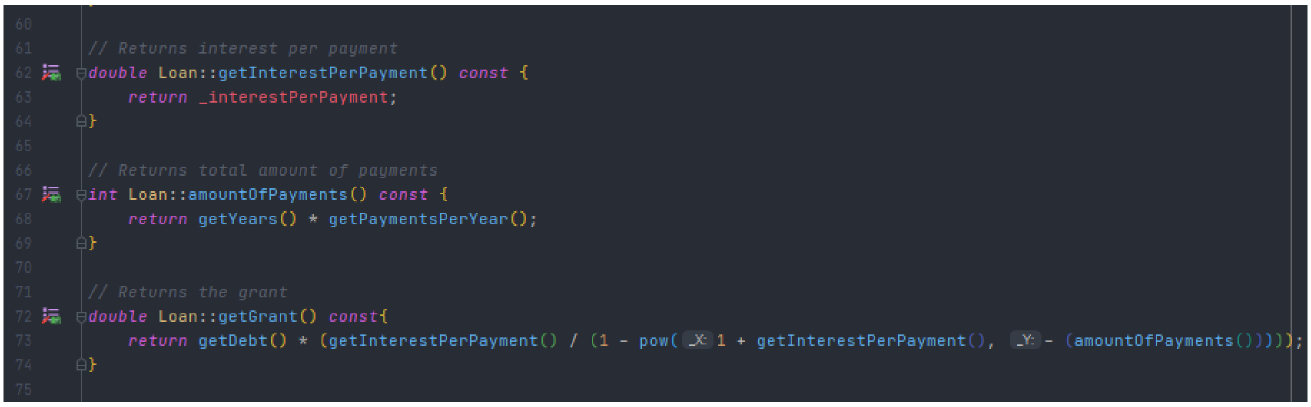
Loan.cpp kildekode



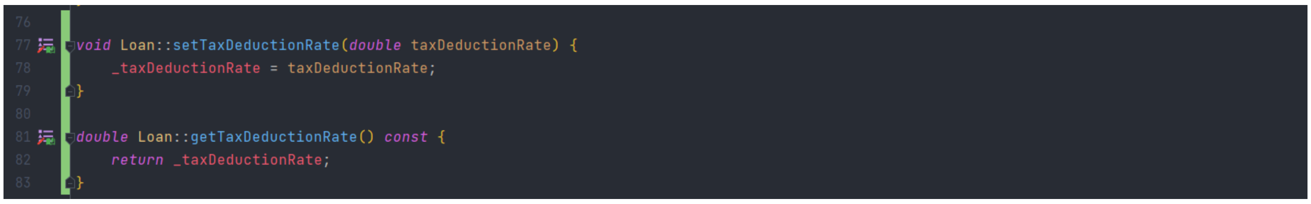
```cpp
#include "Loan.h"
#include <cmath>
#include <iostream>
#include <iomanip>
#include <string>

Loan::Loan() = default; // Default constructor

Loan::Loan (double debt, int years, int paymentsPerYear, double interestRate) { // Main constructor
    _debt = debt;
    _years = years;
    _paymentsPerYear = paymentsPerYear;
    _interestRate = interestRate;
    _interestPerPayment = interestRate/paymentsPerYear;
}

// Returns the number of years the loan lasts
int Loan::getYears () const {
    return _years;
}

// Sets years
void Loan::setYears(int years) {
    _years = years;
}

// Amount of payments per year
int Loan::getPaymentsPerYear() const {
    return _paymentsPerYear;
}

// Sets payments per year
void Loan::setPaymentsPerYear(int paymentsPerYear) {
    _paymentsPerYear = paymentsPerYear;
}
```



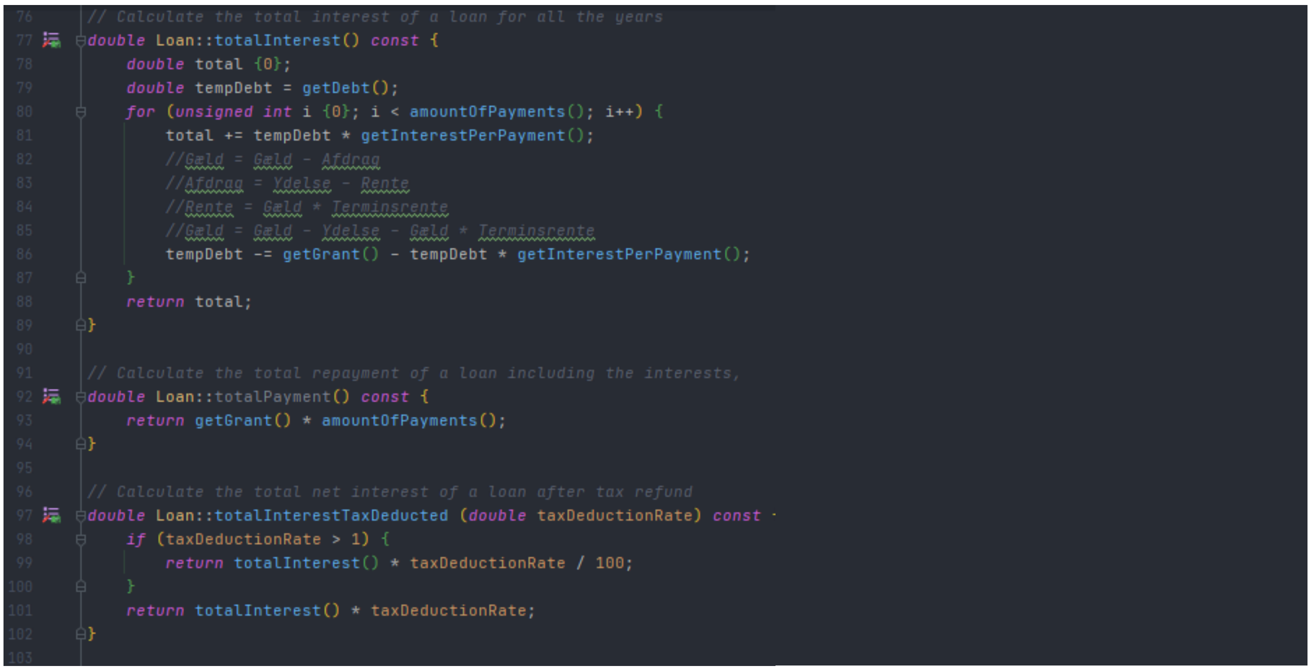
```cpp
// Debt
double Loan::getDebt() const {
    return _debt;
}

// Sets debt
void Loan::setDebt(double debt) {
    _debt = debt;
}

// Returns interest
double Loan::getInterestRate() const {
    return _interestRate;
}

// Sets interest
void Loan::setInterestRate(double rate) {
    _interestRate = rate;
}
```



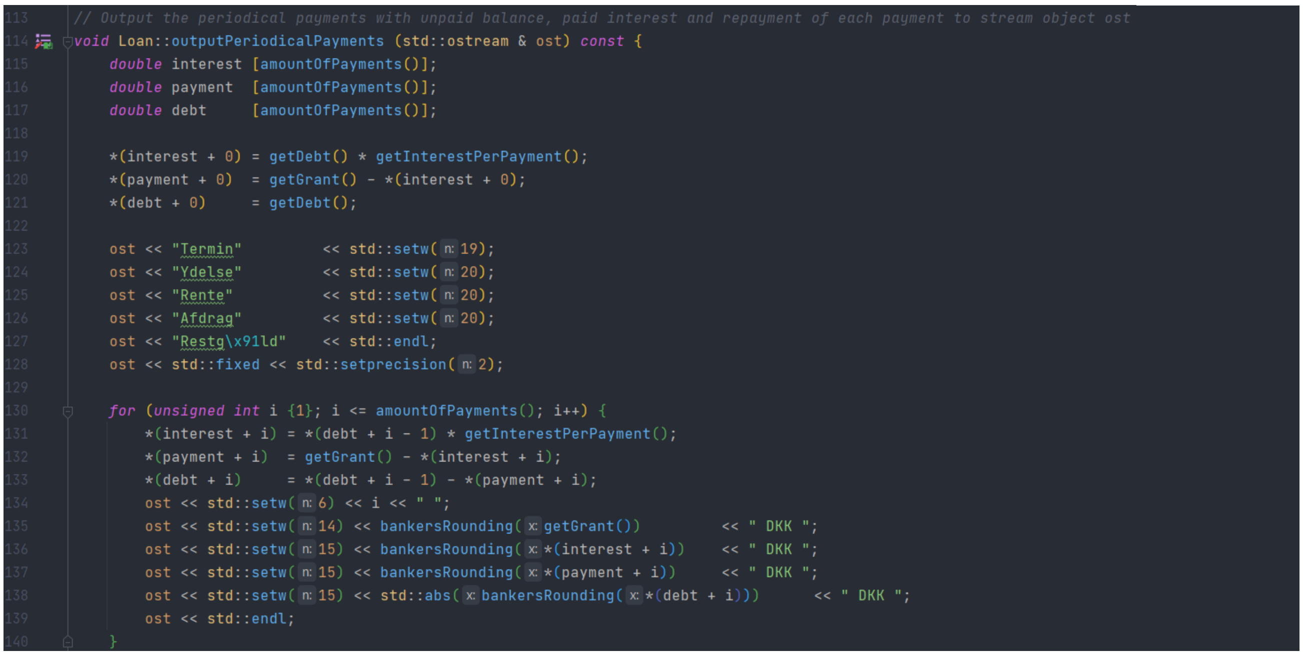
```cpp

// Returns interest per payment
double Loan::getInterestPerPayment() const {
    return _interestPerPayment;
}

// Returns total amount of payments
int Loan::amountOfPayments() const {
    return getYears() * getPaymentsPerYear();
}

// Returns the grant
double Loan::getGrant() const{
    return getDebt() * (getInterestPerPayment() / (1 - pow( X: 1 + getInterestPerPayment(),  Y: - (amountOfPayments()))));
}

```



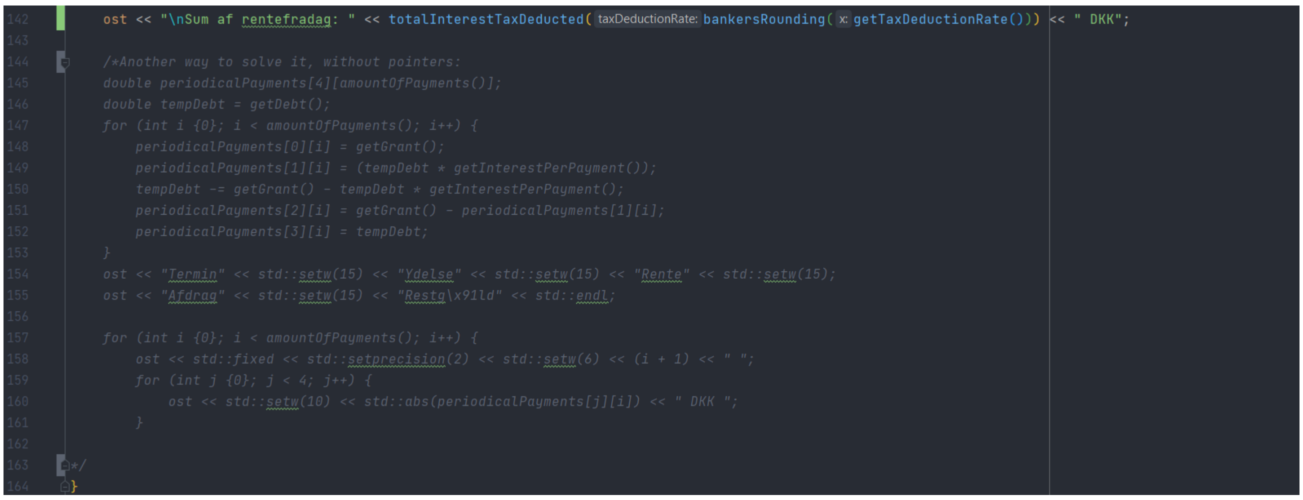
```cpp

void Loan::setTaxDeductionRate(double taxDeductionRate) {
    _taxDeductionRate = taxDeductionRate;
}

double Loan::getTaxDeductionRate() const {
    return _taxDeductionRate;
}
```



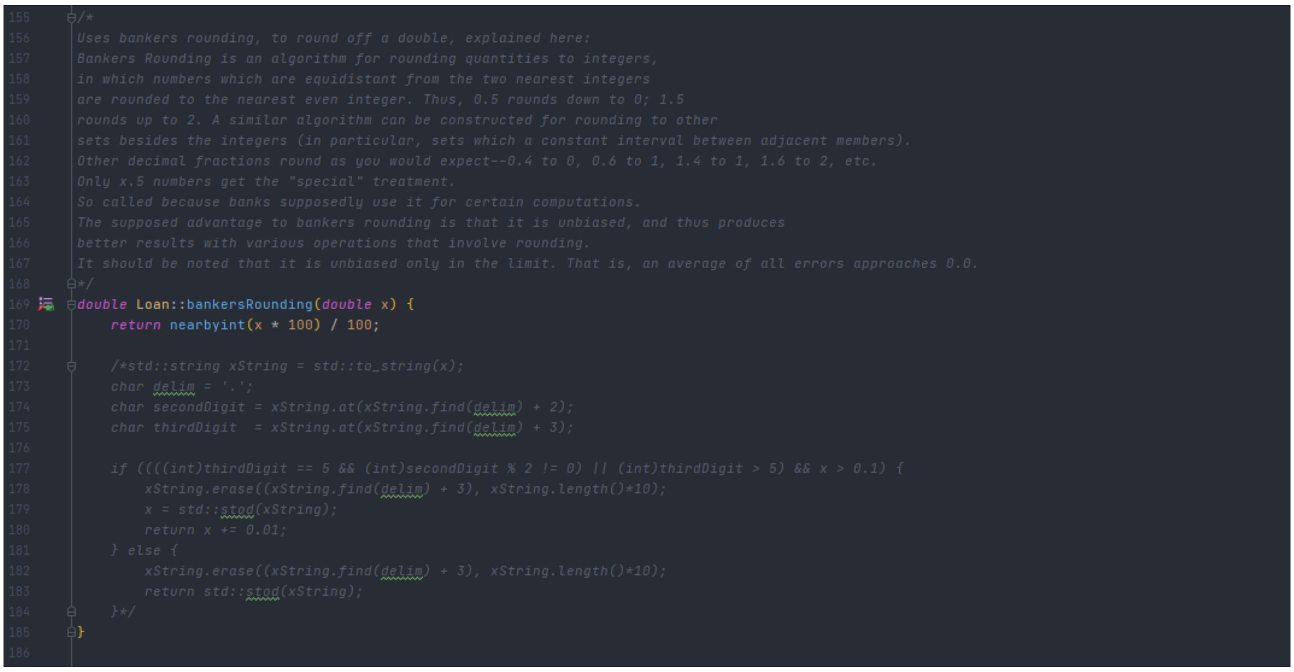
```cpp
// Calculate the total interest of a loan for all the years
double Loan::totalInterest() const {
    double total {0};
    double tempDebt = getDebt();
    for (unsigned int i {0}; i < amountOfPayments(); i++) {
        total += tempDebt * getInterestPerPayment();
        //Gæld = Gæld - Afdrag
        //Afdrag = Ydelse - Rente
        //Rente = Gæld * Terminsrente
        //Gæld = Gæld - Ydelse - Gæld * Terminsrente
        tempDebt -= getGrant() - tempDebt * getInterestPerPayment();
    }
    return total;
}

// Calculate the total repayment of a loan including the interests,
double Loan::totalPayment() const {
    return getGrant() * amountOfPayments();
}

// Calculate the total net interest of a loan after tax refund
double Loan::totalInterestTaxDeducted (double taxDeductionRate) const ·
    if (taxDeductionRate > 1) {
        return totalInterest() * taxDeductionRate / 100;
    }
    return totalInterest() * taxDeductionRate;
}

```



```cpp
// Output the periodical payments with unpaid balance, paid interest and repayment of each payment to stream object ost
void Loan::outputPeriodicalPayments (std::ostream & ost) const {
    double interest [amountOfPayments()];
    double payment  [amountOfPayments()];
    double debt     [amountOfPayments()];

    *(interest + 0) = getDebt() * getInterestPerPayment();
    *(payment + 0)  = getGrant() - *(interest + 0);
    *(debt + 0)     = getDebt();

    ost << "Termin"         << std::setw( n: 19);
    ost << "Ydelse"         << std::setw( n: 20);
    ost << "Rente"          << std::setw( n: 20);
    ost << "Afdrag"         << std::setw( n: 20);
    ost << "Restg\x91ld"    << std::endl;
    ost << std::fixed << std::setprecision( n: 2);

    for (unsigned int i {1}; i <= amountOfPayments(); i++) {
        *(interest + i) = *(debt + i - 1) * getInterestPerPayment();
        *(payment + i)  = getGrant() - *(interest + i);
        *(debt + i)     = *(debt + i - 1) - *(payment + i);
        ost << std::setw( n: 6) << i << " ";
        ost << std::setw( n: 14) << bankersRounding( x: getGrant())        << " DKK ";
        ost << std::setw( n: 15) << bankersRounding( x: *(interest + i))   << " DKK ";
        ost << std::setw( n: 15) << bankersRounding( x: *(payment + i))    << " DKK ";
        ost << std::setw( n: 15) << std::abs( x: bankersRounding( x: *(debt + i)))      << " DKK ";
        ost << std::endl;
    }
```

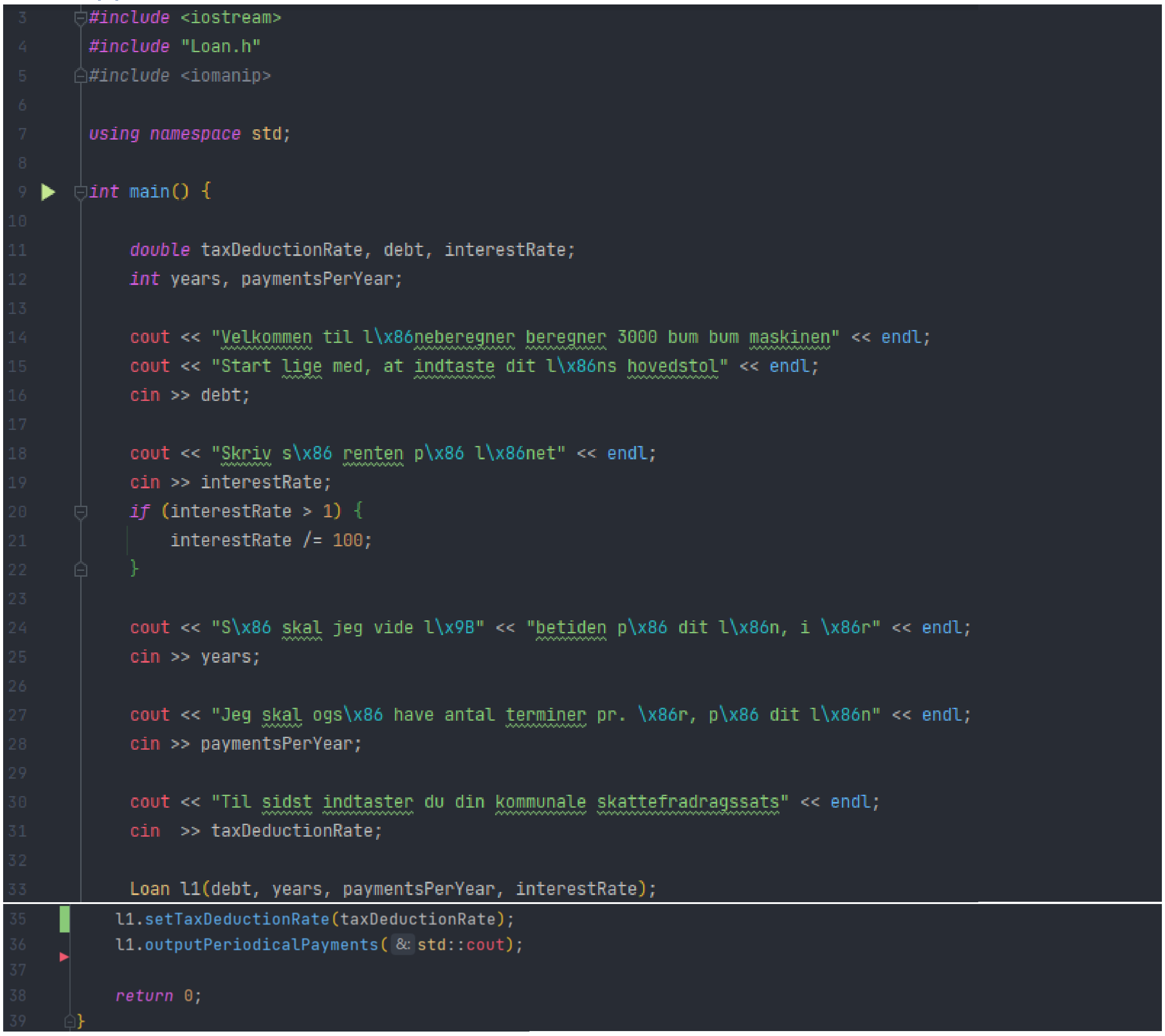


```cpp
    ost << "\nSum af rentefradag: " << totalInterestTaxDeducted( taxDeductionRate: bankersRounding( x: getTaxDeductionRate())) << " DKK";

    /*Another way to solve it, without pointers:
    double periodicalPayments[4][amountOfPayments()];
    double tempDebt = getDebt();
    for (int i {0}; i < amountOfPayments(); i++) {
        periodicalPayments[0][i] = getGrant();
        periodicalPayments[1][i] = (tempDebt * getInterestPerPayment());
        tempDebt -= getGrant() - tempDebt * getInterestPerPayment();
        periodicalPayments[2][i] = getGrant() - periodicalPayments[1][i];
        periodicalPayments[3][i] = tempDebt;
    }
    ost << "Termin" << std::setw(15) << "Ydelse" << std::setw(15) << "Rente" << std::setw(15);
    ost << "Afdrag" << std::setw(15) << "Restg\x91ld" << std::endl;

    for (int i {0}; i < amountOfPayments(); i++) {
        ost << std::fixed << std::setprecision(2) << std::setw(6) << (i + 1) << " ";
        for (int j {0}; j < 4; j++) {
            ost << std::setw(10) << std::abs(periodicalPayments[j][i]) << " DKK ";
        }

*/
}
```



```cpp
/*
Uses bankers rounding, to round off a double, explained here:
Bankers Rounding is an algorithm for rounding quantities to integers,
in which numbers which are equidistant from the two nearest integers
are rounded to the nearest even integer. Thus, 0.5 rounds down to 0; 1.5
rounds up to 2. A similar algorithm can be constructed for rounding to other
sets besides the integers (in particular, sets which a constant interval between adjacent members).
Other decimal fractions round as you would expect--0.4 to 0, 0.6 to 1, 1.4 to 1, 1.6 to 2, etc.
Only x.5 numbers get the "special" treatment.
So called because banks supposedly use it for certain computations.
The supposed advantage to bankers rounding is that it is unbiased, and thus produces
better results with various operations that involve rounding.
It should be noted that it is unbiased only in the limit. That is, an average of all errors approaches 0.0.
*/
double Loan::bankersRounding(double x) {
    return nearbyint(x * 100) / 100;

    /*std::string xString = std::to_string(x);
    char delim = '.';
    char secondDigit = xString.at(xString.find(delim) + 2);
    char thirdDigit  = xString.at(xString.find(delim) + 3);

    if ((((int)thirdDigit == 5 && (int)secondDigit % 2 != 0) || (int)thirdDigit > 5) && x > 0.1) {
        xString.erase((xString.find(delim) + 3), xString.length()*10);
        x = std::stod(xString);
        return x += 0.01;
    } else {
        xString.erase((xString.find(delim) + 3), xString.length()*10);
        return std::stod(xString);
    }*/
}
```

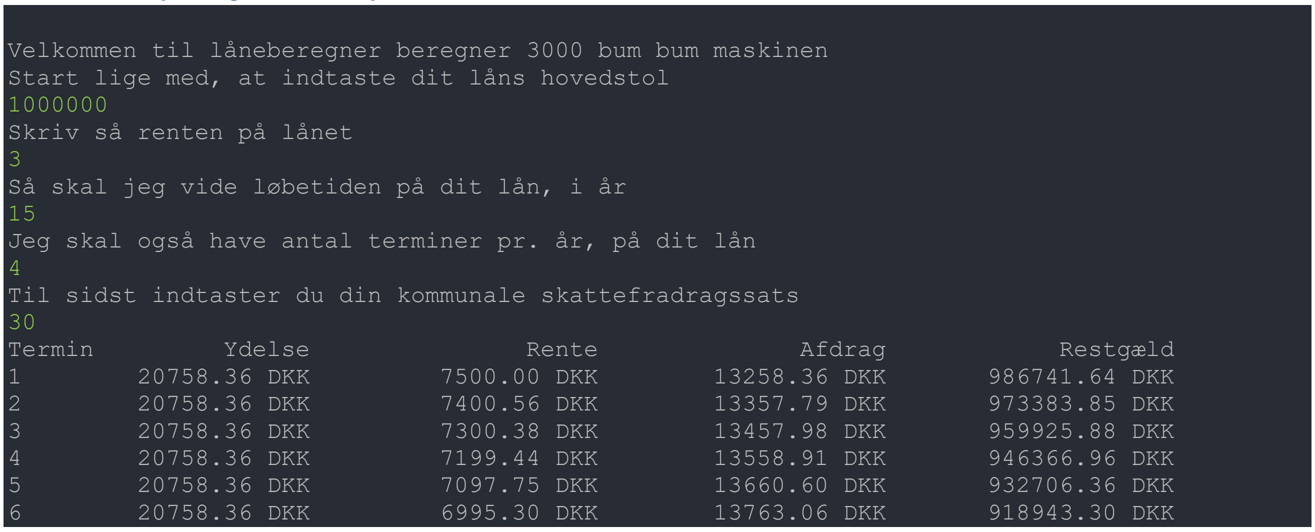
main.cpp kildekode



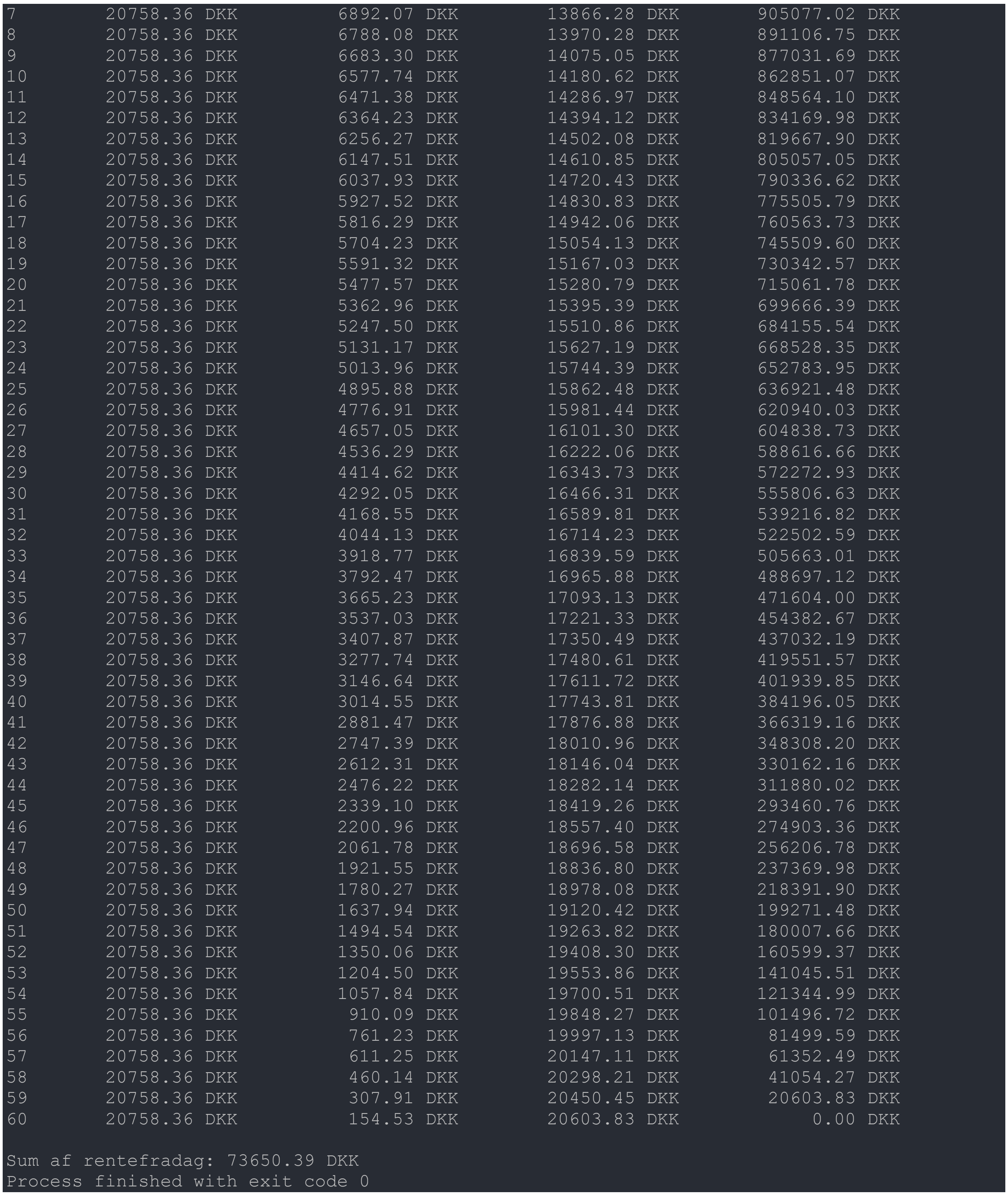
```cpp
#include <iostream>
#include "Loan.h"
#include <iomanip>

using namespace std;

int main() {

    double taxDeductionRate, debt, interestRate;
    int years, paymentsPerYear;

    cout << "Velkommen til l\x86neberegner beregner 3000 bum bum maskinen" << endl;
    cout << "Start lige med, at indtaste dit l\x86ns hovedstol" << endl;
    cin >> debt;

    cout << "Skriv s\x86 renten p\x86 l\x86net" << endl;
    cin >> interestRate;
    if (interestRate > 1) {
        interestRate /= 100;
    }

    cout << "S\x86 skal jeg vide l\x9B" << "betiden p\x86 dit l\x86n, i \x86r" << endl;
    cin >> years;

    cout << "Jeg skal ogs\x86 have antal terminer pr. \x86r, p\x86 dit l\x86n" << endl;
    cin >> paymentsPerYear;

    cout << "Til sidst indtaster du din kommunale skattefradragssats" << endl;
    cin  >> taxDeductionRate;

    Loan l1(debt, years, paymentsPerYear, interestRate);
  l1.setTaxDeductionRate(taxDeductionRate);
  l1.outputPeriodicalPayments( &: std::cout);

  return 0;
}
```

Output fra terminal:

Grå er output, grøn er input



```
Velkommen til låneberegner beregner 3000 bum bum maskinen
Start lige med, at indtaste dit låns hovedstol
1000000
Skriv så renten på lånet
3
Så skal jeg vide løbetiden på dit lån, i år
15
Jeg skal også have antal terminer pr. år, på dit lån
4
Til sidst indtaster du din kommunale skattefradragssats
30
Termin         Ydelse              Rente              Afdrag              Restgæld
1        20758.36 DKK        7500.00 DKK        13258.36 DKK        986741.64 DKK
2        20758.36 DKK        7400.56 DKK        13357.79 DKK        973383.85 DKK
3        20758.36 DKK        7300.38 DKK        13457.98 DKK        959925.88 DKK
4        20758.36 DKK        7199.44 DKK        13558.91 DKK        946366.96 DKK
5        20758.36 DKK        7097.75 DKK        13660.60 DKK        932706.36 DKK
6        20758.36 DKK        6995.30 DKK        13763.06 DKK        918943.30 DKK
```



```
7        20758.36 DKK          6892.07 DKK        13866.28 DKK        905077.02 DKK
8        20758.36 DKK          6788.08 DKK        13970.28 DKK        891106.75 DKK
9        20758.36 DKK          6683.30 DKK        14075.05 DKK        877031.69 DKK
10       20758.36 DKK          6577.74 DKK        14180.62 DKK        862851.07 DKK
11       20758.36 DKK          6471.38 DKK        14286.97 DKK        848564.10 DKK
12       20758.36 DKK          6364.23 DKK        14394.12 DKK        834169.98 DKK
13       20758.36 DKK          6256.27 DKK        14502.08 DKK        819667.90 DKK
14       20758.36 DKK          6147.51 DKK        14610.85 DKK        805057.05 DKK
15       20758.36 DKK          6037.93 DKK        14720.43 DKK        790336.62 DKK
16       20758.36 DKK          5927.52 DKK        14830.83 DKK        775505.79 DKK
17       20758.36 DKK          5816.29 DKK        14942.06 DKK        760563.73 DKK
18       20758.36 DKK          5704.23 DKK        15054.13 DKK        745509.60 DKK
19       20758.36 DKK          5591.32 DKK        15167.03 DKK        730342.57 DKK
20       20758.36 DKK          5477.57 DKK        15280.79 DKK        715061.78 DKK
21       20758.36 DKK          5362.96 DKK        15395.39 DKK        699666.39 DKK
22       20758.36 DKK          5247.50 DKK        15510.86 DKK        684155.54 DKK
23       20758.36 DKK          5131.17 DKK        15627.19 DKK        668528.35 DKK
24       20758.36 DKK          5013.96 DKK        15744.39 DKK        652783.95 DKK
25       20758.36 DKK          4895.88 DKK        15862.48 DKK        636921.48 DKK
26       20758.36 DKK          4776.91 DKK        15981.44 DKK        620940.03 DKK
27       20758.36 DKK          4657.05 DKK        16101.30 DKK        604838.73 DKK
28       20758.36 DKK          4536.29 DKK        16222.06 DKK        588616.66 DKK
29       20758.36 DKK          4414.62 DKK        16343.73 DKK        572272.93 DKK
30       20758.36 DKK          4292.05 DKK        16466.31 DKK        555806.63 DKK
31       20758.36 DKK          4168.55 DKK        16589.81 DKK        539216.82 DKK
32       20758.36 DKK          4044.13 DKK        16714.23 DKK        522502.59 DKK
33       20758.36 DKK          3918.77 DKK        16839.59 DKK        505663.01 DKK
34       20758.36 DKK          3792.47 DKK        16965.88 DKK        488697.12 DKK
35       20758.36 DKK          3665.23 DKK        17093.13 DKK        471604.00 DKK
36       20758.36 DKK          3537.03 DKK        17221.33 DKK        454382.67 DKK
37       20758.36 DKK          3407.87 DKK        17350.49 DKK        437032.19 DKK
38       20758.36 DKK          3277.74 DKK        17480.61 DKK        419551.57 DKK
39       20758.36 DKK          3146.64 DKK        17611.72 DKK        401939.85 DKK
40       20758.36 DKK          3014.55 DKK        17743.81 DKK        384196.05 DKK
41       20758.36 DKK          2881.47 DKK        17876.88 DKK        366319.16 DKK
42       20758.36 DKK          2747.39 DKK        18010.96 DKK        348308.20 DKK
43       20758.36 DKK          2612.31 DKK        18146.04 DKK        330162.16 DKK
44       20758.36 DKK          2476.22 DKK        18282.14 DKK        311880.02 DKK
45       20758.36 DKK          2339.10 DKK        18419.26 DKK        293460.76 DKK
46       20758.36 DKK          2200.96 DKK        18557.40 DKK        274903.36 DKK
47       20758.36 DKK          2061.78 DKK        18696.58 DKK        256206.78 DKK
48       20758.36 DKK          1921.55 DKK        18836.80 DKK        237369.98 DKK
49       20758.36 DKK          1780.27 DKK        18978.08 DKK        218391.90 DKK
50       20758.36 DKK          1637.94 DKK        19120.42 DKK        199271.48 DKK
51       20758.36 DKK          1494.54 DKK        19263.82 DKK        180007.66 DKK
52       20758.36 DKK          1350.06 DKK        19408.30 DKK        160599.37 DKK
53       20758.36 DKK          1204.50 DKK        19553.86 DKK        141045.51 DKK
54       20758.36 DKK          1057.84 DKK        19700.51 DKK        121344.99 DKK
55       20758.36 DKK           910.09 DKK        19848.27 DKK        101496.72 DKK
56       20758.36 DKK           761.23 DKK        19997.13 DKK         81499.59 DKK
57       20758.36 DKK           611.25 DKK        20147.11 DKK         61352.49 DKK
58       20758.36 DKK           460.14 DKK        20298.21 DKK         41054.27 DKK
59       20758.36 DKK           307.91 DKK        20450.45 DKK         20603.83 DKK
60       20758.36 DKK           154.53 DKK        20603.83 DKK             0.00 DKK

Sum af rentefradag: 73650.39 DKK
Process finished with exit code 0
```

Alle tre filer kan findes i .zip filen, hvis programmet ønskes at blive testet.