CS100 Homework 3 (Fall Semester 2018)

Due time: 11:59 pm, November 18, 2018
Homework 3 comprises 4 problems. Please finish all of them independently. Please submit through gitlab and also use OJ to test your code against a standard compiler and standard test cases.

Homework 3 / Problem 1

A structure is defined to represent an arithmetic expression:

```
typedef struct {
      float operand1, operand2;
      char operator; /* operator '+', '-', '*' or '/' */
} bexpression;
```

Write a C function that computes the value of an expression and returns the result. For example, the function will return the value of 4/2 if in the structure passed to it, operand1 is 4, operator is '/' and operand2 is 2. The function prototype is given as:
`float compute1(bexpression expr);`

Write another C function that performs the same computation with the following function prototype:
`float compute2(bexpression *expr);`

Write a C program to test the functions. A sample input and output session is given below:

```
Enter expression (op1 op2 op) :
4 8 +
compute1 = 12.000000
compute2 = 12.000000

Continue ('y' or 'n') :
y
Enter expression (op1 op2 op) :
8 4 /
compute1 = 2.000000
compute2 = 2.000000

Continue ('y' or 'n') :
y
Enter expression (op1 op2 op) :
4 8 *
compute1 = 32.000000
compute2 = 32.000000

Continue ('y' or 'n') :
n
```

A sample template for testing the functions is given below:

```
#include <stdio.h>

typedef struct {
    float operand1, operand2;
    char operator;
} bexpression;

float compute1(bexpression expr);
float compute2(bexpression *expr);

int main() {
  bexpression e;
  char repeat = 'y';

  do {
    printf("Enter expression (op1 op2 op) :\n");
    scanf("%f %f %c", &e.operand1, &e.operand2, &e.operator);
    printf("compute1 = %f\n", compute1(e));
    printf("compute2 = %f\n", compute2(&e));
    getchar();
    printf("\nContinue ('y' or 'n') :\n");
    scanf("%c", &repeat);
  } while (repeat == 'y');
  return 0;
}

float compute1(bexpression expr) {

  /* Add your code here */

}

float compute2(bexpression *expr) {

  /* add your code here */

}
```

Please save your program code in a source file named “hw3_1.c”.

Homework 3/ Problem 2

Write a *recursive* C function **rsquare()** that returns the square of a positive integer number *n*, by computing the sum of odd integers 1, 3, …, (2*n* − 1). The result is passed to the calling function through a pointer variable *result*. For example, if *n* = 4, then *result = 1 + 3 + 5 + 7 = 16 = 4^2; if *n* = 5, then *result = 1 + 3 + 5 + 7 + 9 = 25 = 5^2. The function prototype is:

```
void rsquare(int n, int *result);
```

Write a C program to test the function.

A sample input and output session is given below (the red and italic number is user's input):

```
Enter a number:
4
Result: 16
```

A sample template for the program is given below:

```
#include <stdio.h>
void rsquare(int, int *);

int main() {
  int x, result;

  printf("Enter a number:\n");
  scanf("%d", &x);
  rsquare(x, &result);
  printf("Result: %d\n", result);
  return 0;
}

void rsquare(int num, int *result)
{
  /* Add your code here */
}
```

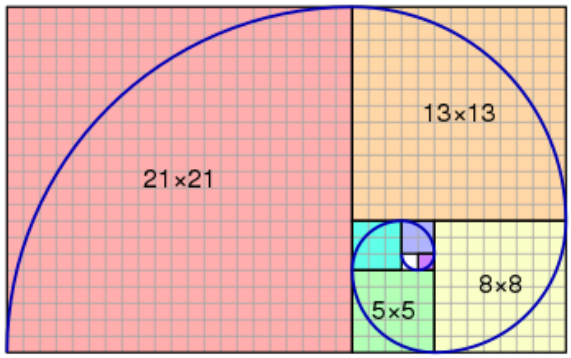
Please save your program code in a source file named "hw3_2.c".

Homework 3 / Problem 3 (Object-oriented programming)

The Fibonacci sequence of numbers is as follows:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, …

It is easily obtained by starting the sequence with the numbers 0 and 1, and then setting each subsequent number as the sum of the two previous ones. The sequence has an interesting application, as a proper arrangement of a sequence of squares of which the side lengths are given by the numbers in the Fibonacci sequence may serve as an approximation of the golden spiral.



[source: https://en.wikipedia.org/wiki/Fibonacci_number]

The task of this problem is straightforward: Implement a Fibonacci number-generator class. The class should be called **`FibonacciGenerator`**, and it should have a procedure called **`void PrintAndAdvance()`**, which will simply print the next number to the console in the format:

```
The next number is 0
The next number is 1
The next number is 1
…
```

Make sure that the number generator has a low memory footprint independently of how many times **`PrintAndAdvance`** is called! Furthermore, the Fibonacci number-generator can be generalized to further sequences by starting sequence with different numbers:

1, 3, 4, 7, 11, 18, 29, 47, …

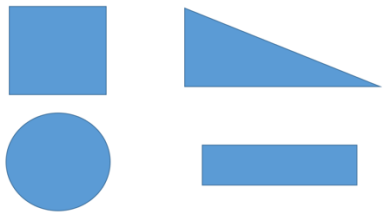
Make sure that the generator can be used for any sequence by providing the first two numbers to the constructor. Also make sure that the generator defaults to the standard Fibonacci sequence if no input parameters are provided to the constructor.

The main function with the I/O functionality is already implemented, it outputs 10 numbers of the standard sequence, followed by 10 numbers of a custom sequence. Your task is to make sure that your implementation of **`FibonacciGenerator`** is compliant with the interface expected by the main procedure and the above specified format of the output. It is

not permitted to change the main procedure! The template code looks as below. Please save your program in a file named “hw3_3.cpp”

```
#include <stdlib.h>
#include <iostream>

/*********************************************************
 *       Declaration of class FibonacciGenerator       *
 *********************************************************/

class FibonacciGenerator {
private:
  //
  // ...
  //

public:

  //
  // ...
  //

  void PrintAndAdvance();
};

/*****************************************************************
 *       Definition of members of class FibonacciGenerator       *
 *****************************************************************/

//
// ...
//

/*************************************************************************
 *  Main Function which is using FibonacciGenerator (do not change!)    *
 *************************************************************************/

int main() {
  FibonacciGenerator fg1;
  for( int i = 0; i < 10; i++ )
    fg1.PrintAndAdvance();
  std::cout << "\n";

  int number1;
  int number2;
  std::cin >> number1 >> number2;
  FibonacciGenerator fg2(number1, number2);
  for( int i = 0; i < 10; i++ )
    fg2.PrintAndAdvance();
  std::cout << "\n";

  return 0;
}
```

Homework 3 / Problem 4 (Inheritance & Polymorphism)



Problem 4 consists of implementing an abstract class **`Shape`** from which the concrete child classes **`Square`**, **`Rectangle`**, **`Triangle`**, and **`Circle`** are derived. Shape should have procedures that permit the output of the area, the perimeter, and the number of corners onto the console. These procedures should be called **`OutputArea`**, **`OutputPerimeter`**, and **`OutputNumberCorners`**. It is not permitted to store the area and the perimeter in member variables of either parent or child class, the values need to be computed just in time! Therefore, every time we call one of the output functions on a shape object, the area or perimeter computation function for this specific shape needs to be invoked.

The interface of **`Shape`** and all child classes again needs to be designed to be compliant with the main procedure, which is already given in advance. The main procedure first defines a mini-database of shapes which is filled through user interaction. The database simply appears in form of a list:

```
std::list<Shape*> shapeDatabase;
```

The main procedure contains a while loop in which the user is prompted to define shapes. Shapes are entered through the console through one of the following inputs:

```
Square <LENGTH_PARAMETER>
Circle <LENGTH_PARAMETER>
Triangle <LENGTH_PARAMETER_1> <LENGTH_PARAMETER_2>
Rectangle <LENGTH_PARAMETER_1> <LENGTH_PARAMETER_2>
```

`<LENGTH_PARAMETER>` and similar place-holders are to be replaced by floating point numbers. Each time a shape has been entered, the main function creates the corresponding shape object and adds it to the list, for example using the command:

```
shapeDatabase.push_back( new Circle( size1 ) );
```

We will see more about the operator **`new`** in class soon. For now, just be aware that it creates an object of a certain type and returns a pointer to that object. This pointer is then stored in the database. While doing so, the pointer is implicitly converted into a Shape pointer!

Each time a shape has been entered, the user is prompted if he wants to add more shapes or not, by which the user is supposed to reply with either `Y` or `N` (anything other than `Y` will simply be interpreted as `N`).

The program finishes with looping through the list and printing the properties of each shape. The output functions are supposed to be called in the following order and return exactly:

```
Area: <AREA_OF_SHAPE>
Perimeter: <PERIMETER_OF_SHAPE>
Corners: <NUMBER_OF_CORNERS>
```

where <AREA_OF_SHAPE>, <PERIMETER_OF_SHAPE>, and <NUMBER_OF_CORNERS> are to be replaced with the just-in-time computed values. A complete input-output session may look as below. Please save your program in a file named "hw3_4.cpp".

```
Enter a type (Circle, Triangle, Square, or Rectangle) and one
or two size parameters, separated with blank spaces:
Circle 0.5
Do you want to add more shapes? (Enter Y or N)
Y
Enter a type (Circle, Triangle, Square, or Rectangle) and one
or two size parameters, separated with blank spaces:
Rectangle 0.4 0.6
Do you want to add more shapes? (Enter Y or N)
Y
Enter a type (Circle, Triangle, Square, or Rectangle) and one
or two size parameters, separated with blank spaces:
Triangle 0.3 0.2
Do you want to add more shapes? (Enter Y or N)
Y
Enter a type (Circle, Triangle, Square, or Rectangle) and one
or two size parameters, separated with blank spaces:
Square 10
Do you want to add more shapes? (Enter Y or N)
N
Properties of shape 0:
Area: 0.785398
Perimeter: 3.14159
Corners: 0
Properties of shape 1:
Area: 0.24
Perimeter: 2
Corners: 4
Properties of shape 2:
Area: 0.03
Perimeter: 0.860555
Corners: 3
Properties of shape 3:
Area: 100
Perimeter: 40
Corners: 4
```

Implement the classes **Shape**, **Circle**, **Triangle**, **Square**, and **Rectangle** to comply with the above requirements. Add your code to the below template. Again, you are not allowed to change the main procedure.

```
#include <stdlib.h>
#include <iostream>
#include <math.h>
#include <string>
#include <list>

/*************************************************************************
 *         Declaration/Definition of the base-class Shape                *
 *************************************************************************/

class Shape //...

/*************************************************************************
 *         Declaration/Definition of the child-classes                   *
 *************************************************************************/

class Rectangle : public Shape // ...
class Square : public Shape // ...
class Triangle : public Shape // ...
class Circle : public Shape // ...

/*************************************************************************
* Main Function which is creating/reporting database (do not change!)    *
*************************************************************************/

int main() {

  //initialize an empty list of shapes
  std::list<Shape*> shapeDatabase;

  //interact with the user: ask the user to enter shapes one by one
  while(1) {
    //print instructions as to how to enter a shape
    std::cout << "Enter a type (Circle, Triangle, Square, or Rectangle) ";
    std::cout << "and one or two size parameters, ";
    std::cout << "separated with blank spaces:\n";
    float size1;
    float size2;

    //check which shape has been requested and store in the database
    std::string shapeType;
    std::cin >> shapeType;
    if( shapeType == std::string("Circle") ) {
      std::cin >> size1;
      shapeDatabase.push_back( new Circle( size1 ) );
    }
    else if ( shapeType == std::string("Triangle") ) {
      std::cin >> size1 >> size2;
      shapeDatabase.push_back( new Triangle(size1,size2) );
    }
    else if ( shapeType == std::string("Square") ) {
      std::cin >> size1;
      shapeDatabase.push_back( new Square(size1) );
    }
    else if (shapeType == std::string("Rectangle") ) {
```

```
      std::cin >> size1 >> size2;
      shapeDatabase.push_back( new Rectangle(size1,size2) );
    }
    else {
      std::cout << "Unrecognized shape!!\n";
    }

    //check if the user wants to add more shapes
    std::cout << "Do you want to add more shapes? (Enter Y or N)\n";
    std::string answer;
    std::cin >> answer;
    if( answer != std::string("Y") )
      break;
  }

  //iterate through the list and output the area, perimeter,
  //and number of corners of each entered shape
  std::list<Shape *>::iterator it = shapeDatabase.begin();
  int shapeCounter = 0;
  while( it != shapeDatabase.end() )
  {
    std::cout << "Properties of shape " << shapeCounter++ << ":\n";
    (*it)->OutputArea();
    (*it)->OutputPerimeter();
    (*it)->OutputNumberCorners();
    it++;
  }

  return 0;
}
```