Programmering for computerteknologi Hand-in Assignment Exercises

Week 12: Bundling Data and their Functions Together

Written by: Alexander A. Christensen (202205452)

Disclaimer: Due to errors with CMake that neither me, nor the TAs have solved, the test-cases have not been run. Instead, the functions have been manually tested.

The code can be found at https://github.com/Aarhus-University-ECE/assignment-12-A-CHRI

Duration class in C

We wish to create a *Duration* class in C++.

(a) Method: *getDuration*

First we create the class with a simple `getDuration` method, that returns the `time` integer attribute. For this we create a private variable `time` in the header file, and declare the public method. In the source code we then add the code for the method.

```
/* getDuration */
int Duration::getDuration() {
    return time;
}
```

(b) Constructor and Destructor methods

Next we implement the constructor method, as well as the destructor method. The constructor sets the `time` attribute to 0.

```
/* Constructor */
Duration::Duration()
{
    time = 0;
}

/* Destructor */
Duration::~Duration()
{
}
```

(c) Second contructor method

Next we wish to create another constructor method, that takes an integer `t`, which is set as the time attribute. This allows us the initialize the *Duration* class with a certain start time.

```
/* Constructor with attribute t */
Duration::Duration(int t) {
    time = t;
}
```

To the header file, another public method `Duration(int t);` is also added.

(d) Testing the class

Next we want to create another file for testing the class, at the current state. Here we test the constructor for various times, and compare them to the result of the `getDuration` method.

```
Duration* d1 = new Duration(); // Empty
Duration* d2 = new Duration(10); // Test number
Duration* d3 = new Duration(2147483647); // Interger boundary

/* Test cases */
(d1->getDuration() == 0) ? printf("PASSED\n") : printf("FAILED\n");
(d2->getDuration() == 10) ? printf("PASSED\n") : printf("FAILED\n");
(d3->getDuration() == 2147483647) ? printf("PASSED\n") : printf("FAILED\n");
```

(e) Implementing the *tick* method

Next we want to add a public method that increases the time attribute by 1.

```
/* tick method */
void Duration::tick() {
    time += 1;
}
```

We also add a test scenario to the test cases.

```
Duration* d = new Duration();

/* Test cases */
(d->getDuration() == 0) ? printf("PASSED\n") : printf("FAILED\n");
d->tick();
(d->getDuration() == 1) ? printf("PASSED\n") : printf("FAILED\n");
```

(f) Expanding the *tick* method

We want to create another tick method that takes a single argument `t` that increases the time attribute by that amount.

```
/* tick method with attribute t */
void Duration::tick(int t) {
    time += t;
}
```

,

Besides this a test scenario, has been added to the previous test code.

```
d->tick(3);
(d->getDuration() == 4) ? printf("PASSED\n") : printf("FAILED\n");
```

(g) Implementing an alarm

When implementing an alarm we wish to create private attributes, containing information on whether the alarm has been set, and at what time it should go off. For this we create a private integer attribute called `alarm` that contains the tick at which the alarm goes off, and a boolean attribute `alarmHasBeenSet` containing info on whether the alarm has been set.

To utilize these attribute we create a public method `setAlarm` to set the alarm, as well as updating the return-type of `tick` to return a boolean, telling us whether the alarm has gone off or not.

```
/* Constructor */
Duration::Duration()
{
    alarm = 0;
    time = 0;
}

/* tick method */
bool Duration::tick() {
    time += 1;
    if (alarmHasBeenSet) {
        if (time >= alarm) {
            alarm = 0;
            return true;
        }
        else return false;
    }
    else return false;
}

/* setAlarm method */
void Duration::setAlarm(int t) {
    alarmHasBeenSet = true;
    alarm = t;
}
```

The following code has been implemented to test the `alarm` method.

```
d = new Duration();

/* Test cases */
d->setAlarm(2);
d->tick() ? printf("FAILED\n") : printf("PASSED\n");
d->tick() ? printf("PASSED\n") : printf("FAILED\n");
d->tick() ? printf("FAILED\n") : printf("PASSED\n");
```

(h) Preventing misuse

To prevent misuse of the program we implement code to ensure that the user does not enter a past date into the new alarm attribute. We want to make sure that the alarm is always greater than the current time. In case the user enters an alarm value that is less, the alarm is reset, and not enabled.

```
/* setAlarm method */
void Duration::setAlarm(int t) {
    /* Prevent entering a date in the past */
    if (t > time)
    {
        alarmHasBeenSet = true;
        alarm = t;
    }
    else {
        alarmHasBeenSet = false;
        alarm = 0;
    }
}
```

(i) Simplifying the *tick* method

Since we spend a lot of code checking if the alarm is set, and whether the ticks exceed the set alarm, we create a new private boolean method `checkAndUpdateAlarm`, which returns whether the alarm is set, and if the ticks exceed the set alarm.

```
bool Duration::checkAndUpdateAlarm() {
    if (alarmHasBeenSet) {
        if (time >= alarm) {
            alarm = 0;
            return true;
        }
        else return false;
    }
    else return false;
}
```

We can now also modify the `tick` method.

```
/* tick method */
bool Duration::tick() {
    time += 1;
    return checkAndUpdateAlarm();
}
```

Final code

Duration.cpp

```
/*
 * Duration Class
 */

#include "duration.h"

#include <assert.h>

/* Constructor */
Duration::Duration()
{
    alarm = 0;
    time = 0;
}

/* Contructor with attribute t */
Duration::Duration(int t) {
    alarm = 0;
    time = t;
}

/* getDuration method */
int Duration::getDuration() {
    return time;
}

/* tick method */
bool Duration::tick() {
    time += 1;
    return checkAndUpdateAlarm();
}

/* tick method with attribute t */
bool Duration::tick(int t) {
    time += t;
    return checkAndUpdateAlarm();
}

/* setAlarm method */
void Duration::setAlarm(int t) {
    /* Prevent entering a date in the past */
    if (t > time)
    {
        alarmHasBeenSet = true;
        alarm = t;
    }
    else {
        alarmHasBeenSet = false;
        alarm = 0;
    }
}

bool Duration::checkAndUpdateAlarm() {
    if (alarmHasBeenSet) {
        if (time >= alarm) {
            alarm = 0;
            return true;
        }
        else return false;
    }
    else return false;
}

/* Destructor */
Duration::~Duration()
{
}
```

Duration.h

```
/*
 * Duration Class Header file
 */

#ifndef _DURATION_H_
#define _DURATION_H_

class Duration
{
private:
    int time;
    int alarm;
    bool alarmHasBeenSet;

    bool checkAndUpdateAlarm();
public:
    Duration();
    Duration(int t);
    ~Duration();

    /* getDuration */
    int getDuration();

    /* tick */
    bool tick();
    bool tick(int t);

    /* setAlarm */
    void setAlarm(int t);
};

#endif
```

main.cpp

```
#include "duration.h"
```

```

#include <stdio.h>

int main (void) {
    /* Test 1 */
    printf("--- TEST 1 ---\n");
    Duration* d1 = new Duration(); // Empty
    Duration* d2 = new Duration(10); // Test number
    Duration* d3 = new Duration(2147483647); // Interger boundary

    /* Test cases */
    (d1->getDuration() == 0) ? printf("PASSED\n") : printf("FAILED\n");
    (d2->getDuration() == 10) ? printf("PASSED\n") : printf("FAILED\n");
    (d3->getDuration() == 2147483647) ? printf("PASSED\n") : printf("FAILED\n");

    delete d1;
    delete d2;
    delete d3;

    /* Test 2 */
    printf("--- TEST 2 ---\n");
    Duration* d = new Duration();

    /* Test cases */
    d->getDuration() == 0 ? printf("PASSED\n") : printf("FAILED\n");
    d->tick();
    d->getDuration() == 1 ? printf("PASSED\n") : printf("FAILED\n");
    d->tick(3);
    d->getDuration() == 4 ? printf("PASSED\n") : printf("FAILED\n");

    delete d;

    /* Test 3 */
    printf("--- TEST 3 ---\n");
    d = new Duration();

    /* Test cases */
    d->setAlarm(2);
    d->tick() ? printf("FAILED\n") : printf("PASSED\n");
    d->tick() ? printf("PASSED\n") : printf("FAILED\n");
    d->tick() ? printf("FAILED\n") : printf("PASSED\n");

    delete d;

    /* Test 4 */
    printf("--- TEST 4 ---\n");
    d = new Duration(10);

    d->setAlarm(5);
    d->tick() ? printf("FAILED\n") : printf("PASSED\n");
    d->setAlarm(12);
    d->tick() ? printf("PASSED\n") : printf("FAILED\n");

    delete d;
}#include "duration.h"

#include <stdio.h>

int main (void) {
    /* Test 1 */
    printf("--- TEST 1 ---\n");
    Duration* d1 = new Duration(); // Empty
    Duration* d2 = new Duration(10); // Test number
    Duration* d3 = new Duration(2147483647); // Interger boundary

    /* Test cases */
    (d1->getDuration() == 0) ? printf("PASSED\n") : printf("FAILED\n");
    (d2->getDuration() == 10) ? printf("PASSED\n") : printf("FAILED\n");
    (d3->getDuration() == 2147483647) ? printf("PASSED\n") : printf("FAILED\n");

    delete d1;
    delete d2;
    delete d3;

    /* Test 2 */
    printf("--- TEST 2 ---\n");
    Duration* d = new Duration();

    /* Test cases */
    d->getDuration() == 0 ? printf("PASSED\n") : printf("FAILED\n");
    d->tick();
    d->getDuration() == 1 ? printf("PASSED\n") : printf("FAILED\n");
    d->tick(3);
    d->getDuration() == 4 ? printf("PASSED\n") : printf("FAILED\n");

    delete d;

    /* Test 3 */
    printf("--- TEST 3 ---\n");
    d = new Duration();

    /* Test cases */
    d->setAlarm(2);
    d->tick() ? printf("FAILED\n") : printf("PASSED\n");
    d->tick() ? printf("PASSED\n") : printf("FAILED\n");
    d->tick() ? printf("FAILED\n") : printf("PASSED\n");

    delete d;

    /* Test 4 */
    printf("--- TEST 4 ---\n");
    d = new Duration(10);

    d->setAlarm(5);
    d->tick() ? printf("FAILED\n") : printf("PASSED\n");
    d->setAlarm(12);
    d->tick() ? printf("PASSED\n") : printf("FAILED\n");

    delete d;
}
```