UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Faculty of EBIT/ Fakulteit van IBIT
Department of Computer Science/ Department van Rekenaarwetenskap
Program Design: Introduction / Programmeringontwerp: Inleiding
COS 110

Assignment 1/ Taak 1
***Deadline**: 2016/09/13 at 23h00 / **Sperdatum**: 2016/09/13 om 23h00*

Instructions/*Instruksies*

1. This assignment must be completed individually./ *Hierdie taak moet individueel voltooi word.*
2. This assignment counts out of 60 marks. */Hierdie taak tel uit 60 punte.*
3. Be ready to upload your assignment well before the deadline as no extension will be granted. */Wees gereed om jou taak lank voor die sluitingsdatum op te laai aangesien geen uitstel toegestaan sal word nie.*
4. If your code does not compile you will be awarded a mark of 0. The output of your program will be primarily considered for marks, although internal structure may also be tested (eg. the presence of certain functions or classes). */As jou kode nie kompileer nie sal jy 'n punt van nul kry. Die uitvoer van jou program word hoofsaaklik oorweeg vir punte, maar interne struktuur mag ook getoets word (bv. of sekere funsies of klasse wel bestaan).*
5. Read the entire assignment thoroughly before you start coding. */Lees deeglik deur die hele taak voor jy begin kodeer.*
6. To ensure that you did not plagiarize, your code **will** be inspected with the help of dedicated software. */Om seker te maak dat geen plagiaat gepleeg word nie, **sal** jou kode geïnspekteer word deur gespesialiseerde sagteware.*
7. Note that plagiarism is considered a very serious offence. Plagiarism will not be tolerated and disciplinary action will be taken against offending students. Please refer to the University of Pretoria's plagiarism page at `http://www.ais.up.ac.za/plagiarism/index.htm`. */Neem kennis dat plagiaat as 'n ernstige oortreding beskou word. Plagiaat word nie geduld nie en dissiplinêre stappe sal geneem word teen oortreders. Verwys asb. na die Universiteit van Pretoria se plagiaat bladsy by `http://www.ais.up.ac.za/plagiarism/index.htm`.*

1 Overview

This assignment focusses on classes and operator overloading. The assignment is out of 70 marks in total.

2 Classes

This section discusses the various classes of the assignment that will be required to complete the tasks discussed in Section 3. Each of these classes' header files are included in the code provided for the assignment. You are **NOT ALLOWED TO CHANGE THE FUNCTIONS PROVIDED**. However, you are allowed to change the header file to add helper functions and other functions as specified in the tasks.

2.1 Wizard

This is the `Wizard` class.

```
#ifndef WIZARD_H
#define WIZARD_H

#include "Spell.h"

class Wizard{
 private:
    Spell* spells;
    int numberOfSpells;
    int maxNumberOfSpells;
    bool hasBeenSummoned;
    int age;
    int numberOfLossedSpells;
    bool hasCompletedTraining;
 public:
    Wizard();
    Wizard(const Wizard& w);
    ~Wizard();
    void addSpell(const Spell& s);
    void deleteSpell(string name);
    int getNumberOfSpells() const;
    void setMaxNumberOfSpells(int m);
    int getMaxNumberOfSpells() const;
    void setAge(int a);
    int getAge() const;
    int getNumberOfLossedSpells() const;
    Spell& getSpell(int index) const;
    bool getHasCompletedTraining() const;
    bool getHasBeenSummoned() const;
};

#endif  /* WIZARD_H */
```

The variables of the class are as follows:

- `spells` which is a dynamically allocated one-dimensional array of `Spell`s.
- `numberOfSpells` which indicates the number of spells that the wizard has mastered.
- `maxNumberOfSpells` which indicates the maximum number of spells that the wizard can master.
- `hasBeenSummoned` that indicates whether the wizard has been summoned to a meeting.
- `age` that indicates the age of the wizard.
- `numberOfLossedSpells` that indicates the number of spells that the wizard cannot cast anymore, i.e. he lost the skill to cast the spell.
- `hasCompletedTraining` that indicates whether a wizard has completed new training.

The functions of the class are as follows:

- `Wizard()`: Default constructor. Here you must set the default values of the variables as follows:
  - `spells`: the size of `spells` is `maxNumberOfSpells`. Each `Spell`'s name should be set to an empty string, i.e. " " (no spaces)
  - `numberOfSpells`: set it to zero
  - `maxNumberOfSpells`: set it to 10
  - `hasBeenSummoned`: set it to `false`
  - `age`: set it to 20
  - `numberOfLossedSpells`: set it to zero
  - `hasCompletedTraining`: set it to `false`
- `Wizard(const Wizard& w)`: copy constructor.
- Destructor: Make sure that you de-allocate any dynamically allocated memory.
- `addSpell(const Spell& s)`: function to add a `Spell` to `spells`. If the `numberOfSpells` has not reached the `maxNumberOfSpells` yet, the `Spell` should be added to the first slot in `spells` where the `Spell`'s `name` is an empty string. If the `maxNumberOfSpells` has been reached, the array should be resized and the new `Spell` should be added to the end of the newly created array. Remember to update the value of `numberOfSpells` and if you have resized the array, also the value of `maxNumberOfSpells`.
- `deleteSpell(string name)`: function to remove a `Spell` from the array of `spells`. Remember to update the value of `numberOfSpells` and to set the values of the deleted value in the array to the values specified for the default constructor. In addition, remember to update `numberOfLossedSpells`.
- `getNumberOfSpells() const`: returns the number of spells that the wizard has mastered.
- `setMaxNumberOfSpells(int m)`: sets the maximum number of spells that the wizard can master.
- `getMaxNumberOfSpells() const`: returns the maximum number of spells that the wizard can master.
- `setAge(int a)`: sets the wizard's age.
- `getAge() const`: returns the wizard's age.
- `getNumberOfLossedSpells() const`: returns the number of spells that the wizard cannot cast anymore.
- `getSpell(int index) const`: returns the `Spell` that is at the specified index of `spells`.
- `getHasBeenSummoned() const`: returns the value of the `hasBeenSummoned` variable.

- `getHasCompletedTraining() const`: returns the value of the `hasCompletedTraining` variable.

2.2 Spell

The `Spell` class is declared as follows:

```
#ifndef SPELL_H
#define SPELL_H

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

class Spell {
private:
    string name;
    int difficultyLevel;
    int skillLevel;
public:
    Spell(string name="Unknown", int difficultyLevel=10, int skillLevel=5);
    Spell(const Spell& sp);
    ~Spell();
    void setName(string n);
    string getName() const;
    void setDifficultyLevel(int d);
    int getDifficultyLevel() const;
    void setSkillLevel(int s);
    int getSkillLevel() const;
};

#endif  /* SPELL_H */
```

It has the following variables:

- `name`: indicates the name of the spell.
- `difficultyLevel`: indicates the difficulty level of the spell, i.e. how difficult the spell is to master.
- `skillLevel`: indicates the skill level of the wizard for the specific spell, i.e. how well or poorly the wizard can cast this spell.

The class has the following functions:

- Default constructor and copy constructor.
- Destructor
- Get and set functions. All get functions only return the specific variable's value. All set functions set a specific variable's value. When setting the `skillLevel`, remember to check the bounds, i.e. it must be greater or equal to zero.

2.3 Hobbit

The header of the `Hobbit` class is as follows:

```
#ifndef HOBBIT_H
#define HOBBIT_H

#include "string"
using namespace std;

class Hobbit{
private:
    string name;
public:
    Hobbit();
    ~Hobbit();
    void setName(string n);
    string getName() const;};

#endif  /* HOBBIT_H */
```

The class contains the following variables:

- `name`: indicates the `Hobbit`'s name.

The class has the following functions:

- A default constructor.
- Get and set functions. All get functions only returns the specific variable's value. All set functions set a specific variable's value.

3 Your task

This assignment consists of the implementation of various operators that have to be overloaded to be used in a specific way.

3.1 Task 1

Implement all of the functions in the header files. This will not earn you any marks, but are required for the rest of the assignment.

3.2 Task 2 (20 marks)

This task implements operator overloading for the `Wizard` class.

3.2.1 operator >

Implement the overloaded greater than ('>') operator for the `Wizard` class. If it is used in the following way:

```
Wizard wiz1, wiz2;
cout << (wiz1 > wiz2) << endl;
```

the statements should result in a zero (false) being produced if `wiz2` has more skills for which its `skillLevel` is greater than the `skillLevel` of `wiz1`, than the number of skills for which `wiz1`'s `skillLevel` is greater than the `skillLevel` of `wiz2`. If the opposite is true, it should produce a one (true). This operator is a relational operator and therefore returns a boolean value. Remember that only spells that occur in both Wizard's `spells` should be compared. Hint: keep a counter for each wizard. If a wizard has a higher skill for a spell, increment that wizard's counter. The wizard whose counter has the highest value in the end is the winner. The operator should then return the correct value according to the counter values.

3.2.2 operator <

Implement the overloaded smaller than ('<') operator for the `Wizard` class. If it is used in the following way:

```
Wizard wiz1, wiz2;
cout << (wiz1 < wiz2) << endl;
```

the statements should result in a zero (false) being produced if `wiz1` has more skills for which its `skillLevel` is greater than the `skillLevel` of `wiz2`, than the number of skills for which `wiz2`'s `skillLevel` is greater than the `skillLevel` of `wiz1`. If the opposite is true, it should produce a one (true). This operator is a relational operator and therefore returns a boolean value. Remember that only spells that occur in both Wizard's `spells` should be compared. Hint: keep a counter for each wizard. If a wizard has a higher skill for a spell, increment that wizard's counter. The wizard whose counter has the highest value in the end is the winner. The operator should then return the correct value according to the counter values.

3.2.3 operator +

Overload the plus ('+') operator in such a way that if it is used as:

```
Wizard wiz1;
Spell spell1;
wiz1 + spell1;
```

it should add the `Spell` to the wizard's `spells`, i.e. call the wizard's `addSpell` function.

3.2.4 operator -

Overload the minus ('-') operator in such a way that if it is used as:

```
Wizard wiz1;
Spell spell1;
wiz1 - "aSpell";
```

it should remove all `Spell`s from the wizard's `spells` that has that specific name ("aSpell"), i.e. call the wizard's `deleteSpell` function. Ensure that the `deleteSpell` function sets all the appropriate variables to the correct default values (refer to the description of `deleteSpell`).

3.3 Task 3 (35 marks)

This task implements the overloaded operators for the `Spell` class. You should overload the following operators:

- Assignment operator ('=')
- Pre-increment operator ('++') that pre-increments the `skillLevel` of the specific `Spell`.
- Post-increment operator ('++') that post-increments the `skillLevel` of the specific `Spell`.
- Pre-decrement operator ('--') that pre-decrements the `skillLevel` of the specific `Spell`. Since you check with `setSkillLevel` for bounds (negative values), it will make sense to check for bounds here too. Or you can just call the `setSkillLevel` function with the new value and let `setSKillLevel` do the boundary checking.
- Post-decrement operator ('--') that post-decrements the `skillLevel` of the specific `Spell`. Since you check with `setSkillLevel` for bounds (negative values), it will make sense to check for bounds here too. Or you can just call the `setSkillLevel` function with the new value and let `setSKillLevel` do the boundary checking.
- Minus equals ('− =') operator that deducts a specified value from the `skillLevel` of the specific `Spell`. Since you check with `setSkillLevel` for bounds (negative values), it will make sense to check for bounds here too. Or you can just call the `setSkillLevel` function with the new value and let `setSKillLevel` do the boundary checking.
- ostream operator ('<<').

You should be able to call these functions as follows:

```
Spell spell1, spell2;
spell2 = spell1;
cout << (spell2--) << endl;
cout << (++spell2) << endl;
spell1 -= 3;
cout << spell2 << endl;
```

For the `ostream` operator, the output should be as follows:

- The `Spell`'s `name` in a field width of 30, right justified, with no spaces, tabs or newlines afterwards
- The `Spell`'s `difficultyLevel` in a field width of 5, right justified, with no spaces, tabs or newlines afterwards
- The `Spell`'s `skillLevel` in a field width of 5, right justified, with no spaces, tabs or newlines afterwards

3.4 Task 4 (5 marks)

This task implements the additional functions for the `Hobbit` class.

Implement the following functions:

- `bool hasBeenSummoned(Wizard& w)`: sets whether a wizard has been summoned to a meeting or not. If a wizard has lost the ability of casting 5 spells, a hobbit will summon the wizard to a meeting and will set the wizard's variable `hasBeenSummoned` to `true`. Therefore, this function must determine whether the variable `hasBeenSummoned` should be set to true. The function returns the value of the wizard's variable `hasBeenSummoned`.
- `bool hasCompletedTraining(Wizard& w)`: sets whether a wizard has completed new training (changes a wizard's variable `hasCompletedTraining`). When the wizard has completed the training (i.e. when this function is called), the hobbit resets the wizard's `numberOfLossedSpells` back to zero, the wizard's `hasBeenSummoned` to `false` and the wizard's `hasCompletedTraining` to `true`. The function returns the value of the wizard's `hasCompletedTraining` variable.

- `void dropWizardSpells(Wizard& w)`: the hobbit investigates whether a wizard should drop or remove any spells. The hobbit goes through the wizard's `spells` and if there is a spell with `skillLevel` of zero, it removes that spell from the Wizard's list of spells by calling the `deleteSpell` function of `Wizard`. After checking all the spells, the hobbit calls its `hasBeenSummoned` function.

4 Submission

Once you are satisfied that your program is working, compress all of your code, including a working Makefile, into a single archive (either .tar.gz, .tar.bz2 or .zip) and submit it for marking to the appropriate upload link (Assignment 1's various upload slots) before the deadline. Each task will have its own upload slot. However, all your code should be included in the archive for all tasks. Also remember that you should first complete Task 1 before uploading any code to any of the other tasks.

Your Makefile should compile your code to an executable named `main`.

IMPORTANT: Remember that your **Makefile must have a static flag and a run rule** at the end of your makefile to execute your compiled program. That is:

```
run:
    ./main
```

Also ensure that you have **NO folders or subfolders** in your archive file.