University American College Skopje

Course: Object Programming

Separating interface from implementation

Exercises

Prepared by: Adrijan Božinovski, Ph.D.

Assignment 1

- Create a class **Player**, to represent a futsal player. It has the following fields:
  - Private:
    - character array *name*, of at most 50 characters
    - int *jersey*
  - Public:
    - A default constructor, empty
    - A parameterized constructor, which will take a character array, and set the name to the parameter, and the *jersey* to a random integer from 1 to 99, inclusive
    - an *int getJersey()* function, which will return the player's *jersey*
    - a *void score(int minute)* function, which will display a message that the player with the given *jersey* and *name* has scored a goal in the minute given by the function's parameter
    - a *void info()* function, which will output a text information about the player's *jersey* and *name*
- Put the class and method declarations in a file **Player.h**, whereas the function code should be put in a definition file **Player.cpp**

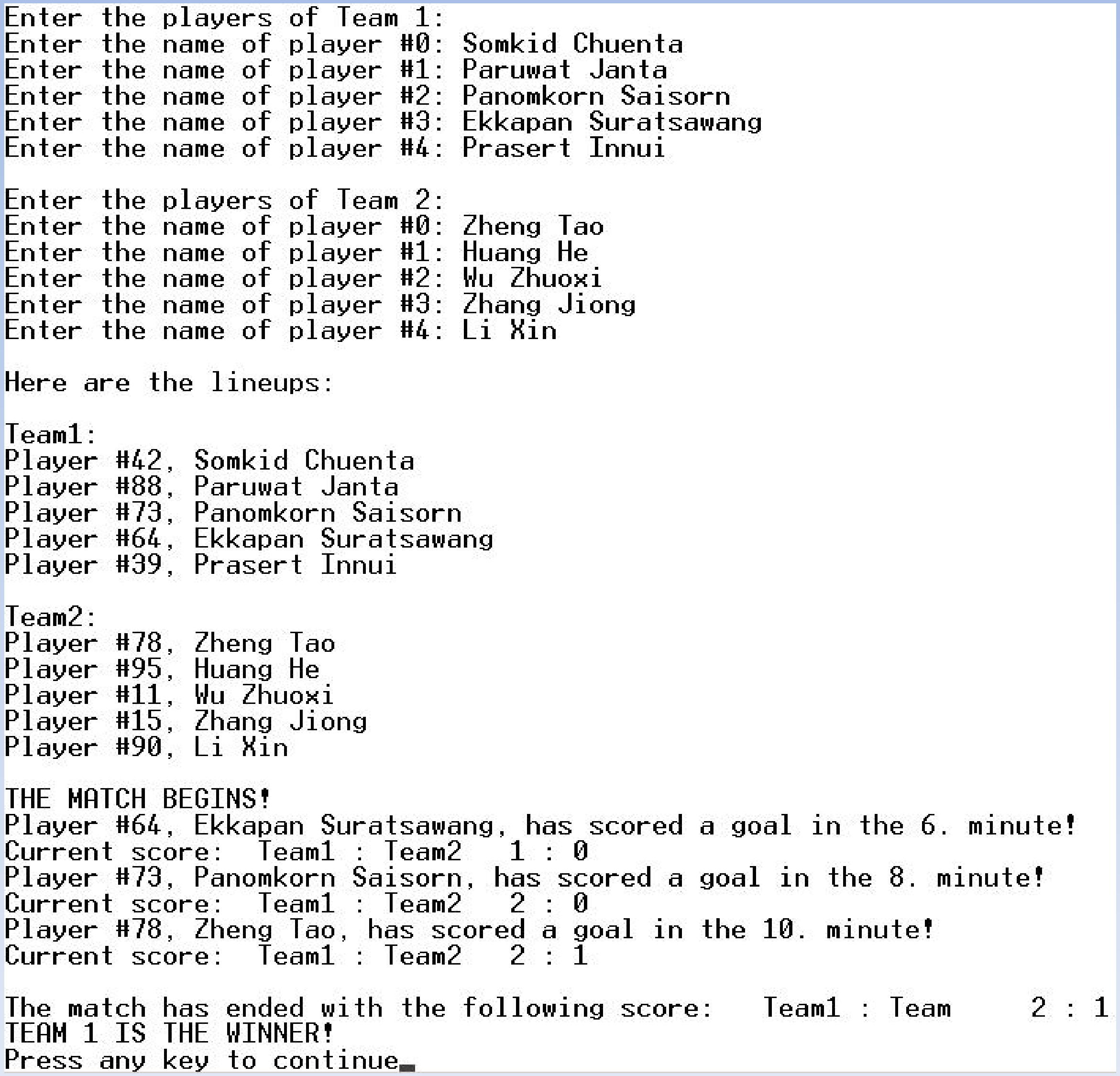
Assignment 1

- Create a driver file **futsal.cpp**, which will contain the main() function and will incorporate the aforementioned **Player** class, along with its method definitions
- In the main() function:
  - randomize the generator of random numbers
  - create an two arrays, each of 5 Player objects
  - create also two integer variables for the score of each team, and another as a counter
  - create an array of 50 characters
  - Input the names of the 5 players of Team 1. Dynamically allocate memory for a new corresponding Player object, and place it in the appropriate place in the appropriate array of players. Perform the necessary memory clean-up after each allocation. Repeat for Team 2.

Assignment 1

- Display the line-ups of both teams, by using the *info()* functions for each of the players on both teams.
- Create a for cycle of 40 trials (minutes). In each cycle
  - select a player that can be the scorer, as a random integer from 0 to 9, inclusive
  - select a *chance value*, as a random integer from 1 to 99, inclusive
  - if the scorer value is from 0 to 4, inclusive, the player that can score is a player with from team 1, with the index of 0 to 4, respectively. If the scorer value is from 5 to 9, inclusive, the player that can score is a player from team 2, with the index of 0 to 4, respectively.
  - the selected player will score a goal if the *chance value* is equal to that player's *jersey* number. If this is the case, invoke that player's *score()* function, and provide the counter (i.e. the current trial, or minute) value as the parameter.
  - if a goal has been scored, increase the score of the appropriate team by one. Also, display the current score of both teams. Do not display anything if a goal has not been scored
- After the 40 minutes, display the final score and declare the winner (the team that has scored more goals), or announce a draw (if the teams have scored an equal amount of goals)

Assignment 1



```
Enter the players of Team 1:
Enter the name of player #0: Somkid Chuenta
Enter the name of player #1: Paruwat Janta
Enter the name of player #2: Panomkorn Saisorn
Enter the name of player #3: Ekkapan Suratsawang
Enter the name of player #4: Prasert Innui

Enter the players of Team 2:
Enter the name of player #0: Zheng Tao
Enter the name of player #1: Huang He
Enter the name of player #2: Wu Zhuoxi
Enter the name of player #3: Zhang Jiong
Enter the name of player #4: Li Xin

Here are the lineups:

Team1:
Player #42, Somkid Chuenta
Player #88, Paruwat Janta
Player #73, Panomkorn Saisorn
Player #64, Ekkapan Suratsawang
Player #39, Prasert Innui

Team2:
Player #78, Zheng Tao
Player #95, Huang He
Player #11, Wu Zhuoxi
Player #15, Zhang Jiong
Player #90, Li Xin

THE MATCH BEGINS!
Player #64, Ekkapan Suratsawang, has scored a goal in the 6. minute!
Current score:  Team1 : Team2   1 : 0
Player #73, Panomkorn Saisorn, has scored a goal in the 8. minute!
Current score:  Team1 : Team2   2 : 0
Player #78, Zheng Tao, has scored a goal in the 10. minute!
Current score:  Team1 : Team2   2 : 1

The match has ended with the following score:   Team1 : Team    2 : 1
TEAM 1 IS THE WINNER!
Press any key to continue_
```

Assignment 2

- Create a header file and a definition file for classes **Cartesian** and **Polar**, enabling also a conversion between them

- This assignment demonstrates linking among many files

Assignment 2

- Create a header file called **Cart.h**, where you'll declare the class **Cartesian**
- In it, forward-declare class **Polar**
- Then, declare class **Cartesian**, with the following members:
  - Private:
    - double x and y
  - Public:
    - get() functions for both fields
    - A default constructor
    - A constructor accepting two double parameters
    - A void print() function
    - A static Cartesian toCart() function, accepting one Polar argument

Assignment 2

- Create a definition file called **Cart.cpp**, where you'll define the methods of the class **Cartesian**
- The get() functions should return their respective values
- The default constructor should initialize both x and y to zero
- The parameterized constructor should initialize x and y to the values of their respective parameters
- The print() function should output the coordinates in the format "(*x*, *y*)"

Assignment 2

- Create a header file called **Polar.h**, where you'll declare the class **Polar**
- In it, forward-declare class **Cartesian**
- Then, declare class **Polar**, with the following members:
  - – Private:
    - double r and f
  - – Public:
    - get() functions for both fields
    - A default constructor
    - A constructor accepting two double parameters
    - A void print() function
    - A static Polar toPolar() function, accepting one Cartesian argument

Why ?

Assignment 2

- Create a definition file called **Polar.cpp**, where you’ll define the methods of the class **Polar**
- The get() functions should return their respective values
- The default constructor should initialize both r and f to zero
- The parameterized constructor should initialize r and f to the values of their respective parameters
- The print() function should output the coordinates in the format “(*r*, *f*)”

Assignment 2

- Create a driver file **CartPolar.cpp**, which will contain the main() function and will incorporate the aforementioned **Cartesian** and **Polar** classes, along with their method definitions

- Also, define the toCart(Polar) function, as returning a Cartesian object, according to the following equation:

$$\mathit{return}\ (x,y), \mathit{where} \qquad \begin{aligned} x &= r \cdot \cos(f) \\ y &= r \cdot \sin(f) \end{aligned}$$

- Also, define the toPolar(Cartesian) function, as returning a Polar object, according to the following equation:

$$\mathit{return}\ (r,f), \mathit{where} \qquad \begin{aligned} r &= \sqrt{x^2 + y^2} \\ f &= \operatorname{atan}(y/x) \end{aligned}$$

Assignment 2

- In the main() function:
  - Create two double variables and input them through the keyboard
  - Create a Cartesian pointer and a Polar pointer
  - Dynamically create a Cartesian object from the two input double values
  - Use that object’s print() function to output it on the screen
  - Assign a Polar object to the Polar pointer from the toPolar() function, using the already created Cartesian object as a parameter
  - Output the newly created Polar object, using its print() function
  - Afterwards, perform memory clean-up

Assignment 2

- – Dynamically create a Polar object from the two input double values
- – Use that object’s print() function to output it on the screen
- – Assign a Cartesian object to the Cartesian pointer from the toCartesian() function, using the already created Polar object as a parameter
- – Output the newly created Cartesian object, using its print() function
- – Afterwards, perform memory clean-up

Assignment 3

- Create a header file **Tower.h**. It should have the declaration of the class **Tower**. The class should have the following items:
  - Public:
    - An integer pointer *gate*, and an integer *numGates*
    - A default constructor
- The definitions of the methods of the class should be placed in the file **Tower.cpp**. They are as follows:
  - The default constructor should generate an array of gates dynamically, and assign that array to the pointer *gate*. The number of gates is random, from 4 to 7. The constructor should display a message about the number of gates in the tower. Each gate has a fortification value (i.e. the value of the element of the array) which is also assigned a random integer value, from 1 to 10

Assignment 3

- Create a header file **Attacker.h**. It should have the declaration of the class **Attacker**. The class should have the following items:
  - Public:
    - An integer *gate*
    - A static integer *numHits*
    - A default constructor
    - A parameterized constructor, taking an integer as the parameter
    - A destructor
- The definitions of the methods of the class should be placed in the file **Attacker.cpp**. They are as follows:
  - The static variable *numHits* should be declared to zero
  - The default constructor should be empty
  - The parameterized constructor should initialize the value *gate*, display a message about which gate (i.e. the value of the parameter) is attacked, and increase *numHits* by one
  - The destructor should display a message that the attacker has been repelled and decrease *numHits* by one

Assignment 3

- Create a driver file **towerDefense.cpp**, which will contain the main() function and will incorporate the aforementioned **Tower** and **Attacker** classes, along with their method definitions

- In the main() function:
  - randomize the generator of random numbers
  - create a Tower object and an Attacker pointer
  - create int variables for *attackLength*, *gateAttacked* and *gateDefended*
  - create int variables for keeping the current system time – a *start* time, a *current* time, and a *moment* time
  - create a bool *breached* variable, to flag whether the tower had been breached
  - count down from 3 to 0, whereas the number should change every second. At the zero moment, display “IMPACT!”
  - set the *attackLength* to be a random integer from 5 to 15 (seconds)
  - set the *start* and *current* time to be equal to the current system time

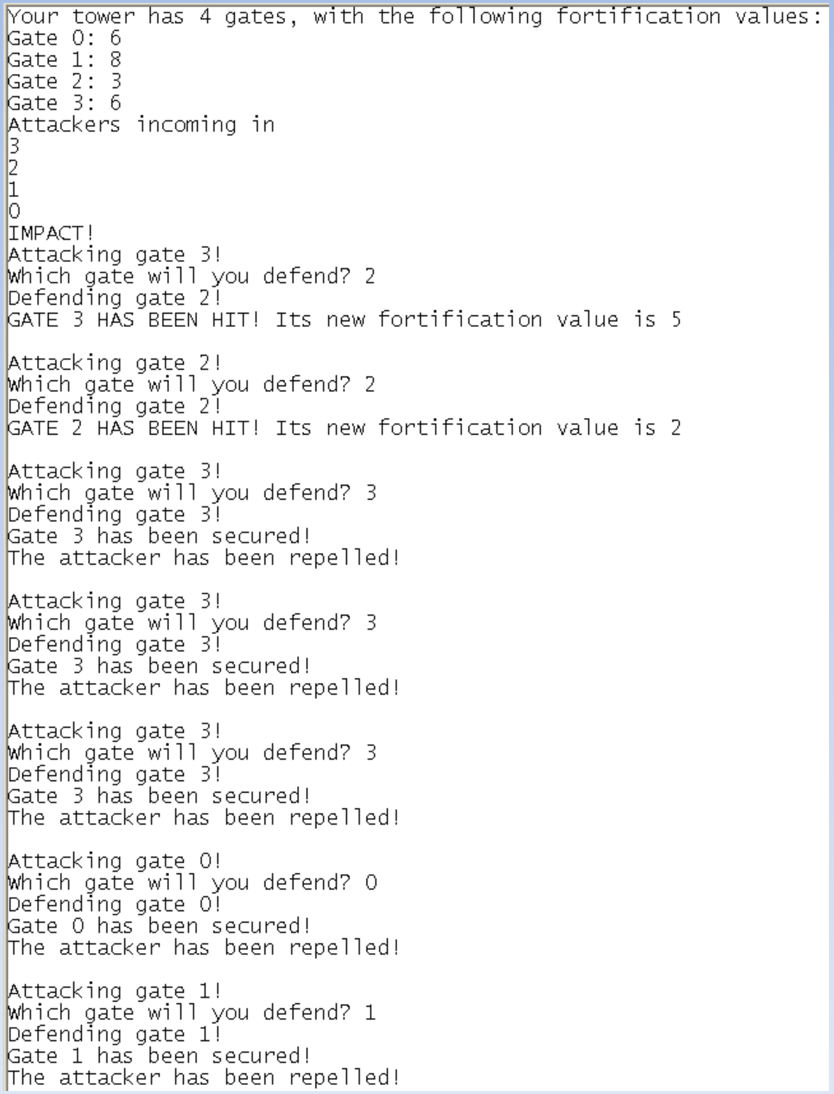
Assignment 3

- while *attackLength* seconds have not yet passed
  - make the *moment* to be equal to the current system time
  - if the attack has just started OR a second has passed
    - make the *current* time equal to the *moment* time
    - select the *gateAttacked* randomly, to be one from the *numGates*
    - dynamically create an Attacker object, with the *gateAttacked* as the parameter
    - ask the user which gate is to be defended. Store the user’s input from the keyboard in the *gateDefended* variable
    - after the user’s input, set the *moment* to be the current system time
    - if the *gateDefended* is equal to the *gateAttacked* AND the *moment* time is equal to the *current* time
      - display a message that the gate has been secured
      - delete the dynamically created Attacker object
    - otherwise
      - decrease the fortification value of the *gateAttacked* gate
      - display a message about it
      - if that gate’s fortification value reaches zero
        - display a message that the corresponding gate has been breached
        - set the *breached* value to true
        - break the while cycle

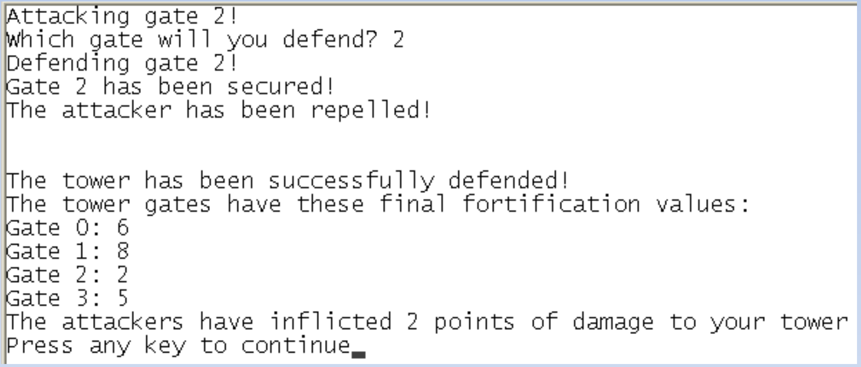
Assignment 3

- – if the gate has been *breached*, display a message that the atackers have overwhelmed the tower
- – otherwise, display a message that the tower has been successfully defended
- – display the final fortification values for each gate
- – display the total *numHits* by the attackers to the tower's gates

Assignment 3



```
Your tower has 4 gates, with the following fortification values:
Gate 0: 6
Gate 1: 8
Gate 2: 3
Gate 3: 6
Attackers incoming in
3
2
1
0
IMPACT!
Attacking gate 3!
Which gate will you defend? 2
Defending gate 2!
GATE 3 HAS BEEN HIT! Its new fortification value is 5

Attacking gate 2!
Which gate will you defend? 2
Defending gate 2!
GATE 2 HAS BEEN HIT! Its new fortification value is 2

Attacking gate 3!
Which gate will you defend? 3
Defending gate 3!
Gate 3 has been secured!
The attacker has been repelled!

Attacking gate 3!
Which gate will you defend? 3
Defending gate 3!
Gate 3 has been secured!
The attacker has been repelled!

Attacking gate 3!
Which gate will you defend? 3
Defending gate 3!
Gate 3 has been secured!
The attacker has been repelled!

Attacking gate 0!
Which gate will you defend? 0
Defending gate 0!
Gate 0 has been secured!
The attacker has been repelled!

Attacking gate 1!
Which gate will you defend? 1
Defending gate 1!
Gate 1 has been secured!
The attacker has been repelled!
```



```
Attacking gate 2!
Which gate will you defend? 2
Defending gate 2!
Gate 2 has been secured!
The attacker has been repelled!


The tower has been successfully defended!
The tower gates have these final fortification values:
Gate 0: 6
Gate 1: 8
Gate 2: 2
Gate 3: 5
The attackers have inflicted 2 points of damage to your tower
Press any key to continue_
```