**CS201R**

**Problem Solving & Programming II**

**Program 5 – Dream Teams**

**Assignment learning objectives:**

Implement inheritance and polymorphism through base and derived classes.

Implement classes and class functions including constructors, accessors, mutators

Read data from a file (.csv) and utilize try/except to error check.

Pointer used to create a vector of parent and child objects.

Create functions to print a menu and search object data for provided search results.

**Base Class**

The base class will be a BasketballPlayer class. From this, you will create two child classes: CollegeBasketballPlayer and ProBasketballPlayer which will extend the BasketballPlayer class.

The BasketballPlayer class will have instance data as listed in the table below:

|  |
| --- |
| Basketball Player  -stl: int  -blk : int  -tov : int  -pts : int  -effRating: float  -value: int |
| -playerType : char  -playerName: string  -games: int  -efg : float  -trb : int  -ast: int |
| +BasketBallPlayer()  +BasketBallPlayer(playerType, playerName, games, efg, trb, ast, stl, blk, tov,pts)  +getPlayerType( ):char  +getPlayerName( ):string  +getGames( ):int  +getEFG( ):float  +getTRB( ):int  +getAST( ):int  +getSTL( ):int  +getBLK( ):int  +getTOV( ):int  +getPoints( ): int  +getEffRating() : float  +getValue() : int  +setPlayerType(char) : void  +setPlayerName(string) : void  +setGames(int) : void  +setEFG(float) : void  +setTRB(int) : void  +setAST(int) : void  +setSTL(int) : void  +setBLK(int) : void  +setTOV(int) : void  +setPoints(int) : void  +setEffRating() \*\*: void  +setValue() \*\*: void  +print() : void |

Note: all attributes should be declared as protected instead of private.

all methods should be declared as public

\*\*see how to set these fields below

|  |
| --- |
| BasketBallPlayerCollege |
| -team: string |
| +setTeam(string) : void  +getTeam() : string  +print() : void |

|  |
| --- |
| BasketBallPlayerPro |
| -years : int  -position : string  -contractValue : int |
| +BasketBallPlayerPro()  +BasketBallPlayerPro(playerType, playerName, games, efg, trb, ast, stl, blk, tov,pts,years,position)  +setYears(int) : void  +getYears() : int  +getPosition() : string  +getContractValue() : int  +setPosition(string) : void  +setContractValue() : void  +print() : void |

NOTE: age is in the input file, but you are not using this data.

**The Process:**

1. Create your base class which includes:
2. Constructors:

* a default constructor (all instance data are initialized as “unknown” (if it is a string) or 0 for numeric data – including the effRating & value)
* a 10-parameter constructor that receives values for each instance data for the base basketball player (type, Player, G, eFG%, TRB, AST, STL, BLK, TOV, PTS) and assigns them all appropriately. This constructor will also make a call to additional methods setEffRating & setValue to set the last two fields.

1. Methods/Class Functions

* Create accessors and mutators for all instance data.
* Create a print method that will make the base class abstract.
* setEffRating will set the player’s effRating using the following equation:

(PTS + TRB + AST + STL + BLK – (PTS/eFG% - PTS + TOV))/G

* setValue will set the player’s value based on the following rules:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  | playerValue |
| eFG% > .70 or effRating > 20 |  |  |  | 10 |
| eFG% > .65 or effRating > 15, |  |  |  | 9 |
| eFG% > .65, stl/g > 2, | pts/g > 20 | trb/g > 5 | TOV/g <= 1 | 9 |
|  |  |  |  |  |
| eFG% > .60 or effRating > 13 |  |  |  | 8 |
| eFG% > .60, blk/g >3, | pts/g > 15 | trb/g > 5 |  | 8 |
| eFG% > .60 | pts/g > 15 | trb/g > 5 | TOV/g < 2 | 8 |
|  |  |  |  |  |
| eFG% > .55 or effRating > 10 |  |  |  | 7 |
| eFG% > .55 | pts/g > 15 | trb/g > 5 |  | 7 |
| eFG% > .55, blk/g > 3 | pts/g > 10 | trb/g > 4 |  | 6 |
| eFG% > .45 | pts/g > 10 | trb/g > 4 |  | 5 |
| eFG% > .40 | pts/g > 5 | trb/g > 3 |  | 3 |
| eFG% > .35 | pts/g > 5 | trb/g > 3 |  | 2 |
|  |  |  |  | 0 |

Verify that your class compiles before continuing.

1. Create the **ProBasketballPlayer** derived class which extends BasketballPlayer.
2. Add 3 new instance data:

* years (yearsInLeague) : int (note: if years < 0, set to 0)
* pos (position) : string
* contractValue : int

1. Constructors:
   * a default constructor (use the parent constructor for all previously defined attributes). Set additional numeric fields to 0 and pos to ‘unknown’.
   * a 13-position constructor that receives all parameters and uses the parent constructor for all previously defined attributes. This constructor will also call the setContractValue method (see below) to set the instance data for contractValue.

c) Methods/Class Functions

* + Add accessors & mutators for the new instance data
  + Add a print which will return the name and the player’s name, playerValue, effRating, contractValue and positiion.
  + Add setContractValue which returns an int value based on the logic in the following table.

|  |  |  |
| --- | --- | --- |
| playerValue | effRating | contractValue |
| 10 |  | $12.000,000 |
| 9 |  | $10,000,000 |
| 8 |  | $7,500,000 |
| 7 |  | $6,000,000 |
| 5,6 |  | $5,000,000 |
| 3,4 |  | $2,500,000 |
| <3 |  | $500,000 |

1. Create the **CollegeBasketballPlayer** derived class which extends BasketballPlayer.
2. Add 3 new instance data: Team (string)
3. Constructors:
   * a default constructor (use the parent constructor for all previously defined attributes). Set additional numeric fields to 0 and pos to ‘unknown’.
   * an 11-parameter constructor that receives 11 parameters and uses the parent constructor for all previously defined attributes.

c) Methods/Class Functions

* + Add accessors & mutators for the new instance data
  + Add a print that will print the player’s name, effRating, and playerValue.

Verify that these compile and fix any errors.

1. Input the two player files and store the players in **ONE** vector.
2. Create a vector that consists of pointers to BBPlayer type.
3. Read the input files and add the pointers to the derived class records to this vector.
4. Print the vector and verify that your code is working by checking the input and printing the vector contents.

1. Create your dream teams. You will create 3 ‘dream’ teams that each consist of 12 players. Each team will need to follow the rules given below:

* 1 All NCAA
  + You pick the best qualified 12 players in the NCAA based on their playerValue. If there is a tie, you would then pick the player with the highest effRating.
  + Table

    Description automatically generatedPrint the team roster.
* 1 Mix of Pro & NCAA
  + You will be picking the top 2 players (in each position) from the professional list and the highest qualified players in the NCAA to make this team.
  + You must pick 2 centers, 2 forward positions and 2 guard positions from the PRO list choosing the highest qualified players for each position. Top players will be chosen based on having the highest playerValue, and then followed by the highest effRating.
  + You will then pick the top 6 players in the NCAA based on their playerValue, followed by their effRating.
  + Print the team roster.

Table

Description automatically generated with low confidence

* 1 All Pro
  + You must pick 2 centers, 5 forward positions and 5 guard positions. To do this, you will need to start by:
* Pick the best center and the best 2 forward and 2 guard positions.
* In order to get a well-rounded team, you realize that the next 7 players must be added in order by importance of position, but also with a maximum playerValue for each ‘round’ so that you do not exceed the maximum payout for this team ($98 million)
  + - * center (max playerValue = 8)
      * forward, guard (max playerValue = 8)
      * guard, forward (max playerValue = 6)
      * forward, guard (max playerValue = 5)
  + Print the team roster along with the total cost for funding this team.

Table

Description automatically generated with medium confidence

1. Errors should be written to another output file. Errors would include any conversion errors and any player where the effRating could not be calculated due to an insufficient number of games played (<=5). Sample output is below:

Text, table

Description automatically generated

**Input Files:**

Input files can be found here:

[playerStatsNCAA.csv](https://umkc.box.com/s/avjnf90zuyfkxu7gkgpzphzr81dus5sq)

[playerStatsPRO.csv](https://umkc.box.com/s/kk3dg13a5yrfwidyj18672cutpqxmw75)

**Submission & Grading:**

I am reviewing the grading rubric and work submission for this program. Please look for information to follow closer to the due date.

Appendix A: Definition of the Fields

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Column | Data Type | Meaning | Position description | Note |
| playerType | char | P = professional  C = college | This will be set based on the file you are currently reading |  |
| Player | string | Player name |  |  |
| G | int | Games played | CFG |  |
| eFG% | float | Effective field goal percent | (FG + .05 \* 3P)/FGA |  |
| TRB | int | total rebounds | C,F ,G |  |
| AST | int | assists | G |  |
| STL | int | steals | G |  |
| BLK | int | blocks | C,F |  |
| TOV | int | turnovers |  |  |
| PTS | int | points | C,F,G |  |
| AGE | Int | Age of player |  | This field is only in PRO data |
| YRS | Int | Years in league | NOTE this is approximate based on age - if <= 0 , assume 1 year | This field is only in PRO data |
| POS | string | Player position  C – center  F - forward  G - guard | C – center  PF – power forward  (forward)  SF – small forward  (forward)  SG – shooting guard  (guard)  PG – point guard  (guard) | This field is only in PRO data |
| TM | String | Team |  | This field is only in NCAA data |