# Assignment Description

This week you will start on an NFL Football Team Program.

For starters, you will create an abstract base class called player. This will have attributes of name and playerID (number), methods getName and getPlayerID, and abstract methods getPlayerPosition, play, and toString (which returns a string with the player name, number, position, etc.)

You will create subclasses (all of which extend from player) for

* Offense
  + QuarterBack
  + Wide Receiver
  + Tight End
  + Running Back
  + Offensive Lineman
* Defense
  + Defensive Lineman
  + Linebacker
  + Defensive Back
* Special Teams
  + Kicker
  + Holder
  + Punter
  + Returner (Kick, Punt)

Create a menu driven program that will allow the user to add players to a team (use only one data structure to hold your players). Add at least one player from each category to your order and then print out the list of players on a team (name, number, position, description, etc.). The PLAY method should indicate how the player participates in the game (e.g. kicks, blocks, throws, etc.). Display the number of players on the team.

# 1 Readme Documentation

This program will make new players for an NFL team. The players will have a name, number, and position associated with them. You can view the team, make new players, and exit program.

# 2 Flowchart Screen Shots

# 3 UML and Use Case Diagrams

# 4 Source Code of All files (.h, .cpp)

#include *<iostream>*

#include *<iomanip>*

#include *<string>*

#include *<cctype>*

#include *<stdexcept>*

#include *<map>*

#include *<vector>*

**using** **namespace** **std**;

*/\**

*Name: Football Class Inheritance*

*Author: Wesley Hixon*

*Date Last Updated: 11/29/2024*

*Purpose: This program will create new football players which inherit from the abstract player class*

*\*/*

*// Abstract player class*

**class** **player**{

**protected**:

string name;

int playerID;

**public**:

player(string n, int id): name(n), playerID(id){};

string getName(){**return** name;}

int getPlayerID(){**return** playerID;}

**virtual** string getPlayerPosition() = 0;

string toString(){

string playerString;

playerString = "Name: " + name + "**\n**Number: " + to\_string(playerID) + "**\n**Position: " + getPlayerPosition() + "**\n**Description: " + play();

**return** playerString;

}

void setName(string inputName){name = inputName;}

void setPlayerID(int inputID){playerID = inputID;}

**virtual** string play() = 0;

};

*// Enum for offensive roles*

**enum** **offensiveRole**{

QUARTERBACK,

WIDERECEIVER,

TIGHTEND,

RUNNINGBACK,

OFFENSIVELINEMAN

};

*// Map which links offensive roles to what they do*

map<offensiveRole, string> offensivePlays{

{QUARTERBACK, "throws"},

{WIDERECEIVER, "catches"},

{TIGHTEND, "blocks and catches"},

{RUNNINGBACK, "runs the ball"},

{OFFENSIVELINEMAN, "protects the quarterback"}

};

*// Derived offense class*

**class** **Offense**: **public** player{

**private**:

offensiveRole position;

**public**:

*// Creates new Offense obj with name, id, and static\_cast(pos)*

Offense(string n, int id, int pos): player(n, id), position(**static\_cast**<offensiveRole>(pos)) {};

*// Returns string for position*

string getPlayerPosition() **override** {

string positionString;

**switch**(position){

**case** QUARTERBACK:

positionString = "Quarterback";

**break**;

**case** WIDERECEIVER:

positionString = "Wide Receiver";

**break**;

**case** TIGHTEND:

positionString = "Tight End";

**break**;

**case** RUNNINGBACK:

positionString = "Running Back";

**break**;

**case** OFFENSIVELINEMAN:

positionString = "Offensive Lineman";

**break**;

}

**return** positionString;

}

*// Returns description of what they do*

string play() **override** {

string playString;

playString = name + " " + offensivePlays[position];

**return** playString;

}

};

*// Enum of defensive roles*

**enum** **defensiveRole**{

DEFENSIVELINEMAN,

LINEBACKER,

DEFENSIVEBACK

};

*// Map of defensive roles and what they do*

map<defensiveRole, string> defensivePlays{

{DEFENSIVELINEMAN, "tackles the ball carrier"},

{LINEBACKER, "blocks a pass"},

{DEFENSIVEBACK, "blocks a pass"}

};

*// Derived defense class*

**class** **Defense**: **public** player{

**private**:

defensiveRole position;

**public**:

*// Constructor that takes name, id, and static\_cast(pos)*

Defense(string n, int id, int pos) : player(n, id), position(**static\_cast**<defensiveRole>(pos)) {};

*// Returns position string*

string getPlayerPosition() **override** {

string positionString;

**switch**(position){

**case** DEFENSIVELINEMAN:

positionString = "Defensive Lineman";

**break**;

**case** LINEBACKER:

positionString = "Linebacker";

**break**;

**case** DEFENSIVEBACK:

positionString = "Defensive Back";

**break**;

}

**return** positionString;

}

*// Returns what they do in the game*

string play() **override** {

string playString;

playString = name + " " + defensivePlays[position];

**return** playString;

}

};

*// Enum for special roles*

**enum** **specialRole**{

KICKER,

HOLDER,

PUNTER,

RETURNER

};

*// Links special roles to what they do*

map<specialRole, string> specialPlays{

{KICKER, "kicks the ball"},

{HOLDER, "holds the ball for the kicker"},

{PUNTER, "punts the ball"},

{RETURNER, "runs the ball"}

};

*// Derived class for special players*

**class** **specialPlayer**: **public** player{

**private**:

specialRole position;

**public**:

*// Constructor that takes name, id, and static\_cast(pos)*

specialPlayer(string n, int id, int pos) : player(n, id), position(**static\_cast**<specialRole>(pos)) {};

string getPlayerPosition() **override** {

string positionString;

**switch**(position){

**case** KICKER:

positionString = "Kicker";

**break**;

**case** HOLDER:

positionString = "Holder";

**break**;

**case** PUNTER:

positionString = "Punter";

**break**;

**case** RETURNER:

positionString = "Returner";

**break**;

}

**return** positionString;

}

string play() **override** {

string playString;

playString = name + " " + specialPlays[position];

**return** playString;

}

};

*// Function prototypes*

int getPlayerID();

int getMenuChoice(int upperBound, int lowerBound);

int main(){

*// Makes a team out of player pointers*

*// Can't make vector out of player because abstract class*

vector<unique\_ptr<player>> team;

*// Running loop*

bool running = true;

**while**(running){

*// Main menu*

cout << endl <<"Please choose from the following options:" << endl;

cout << "1. Create offensive player" << endl;

cout << "2. Create defensive player" << endl;

cout << "3. Create special player" << endl;

cout << "4. Show team" << endl;

cout << "5. Exit Program" << endl;

int userChoice = getMenuChoice(1, 5);

*// Exits program*

**if**(userChoice == 5){

running = false;

**break**;

}

*// Prints team*

**if**(userChoice == 4){

cout << endl << "Team:" << endl;

**for**(**auto** itr = team.begin(); itr != team.end(); itr++){

cout << endl << (\*itr)->toString() << endl;

}

**continue**;

}

*// Otherwise, creates new player*

*// Get player name*

string playerName;

cout << endl << "What is the name of your player?" << endl;

cin.ignore(1000, '\n');

getline(cin, playerName);

*// Get player ID*

int playerID = getPlayerID();

int positionChoice;

*// Makes new offense player*

**if**(userChoice == 1){

cout << endl << "What position does your offensive player play?" << endl;

cout << "1. Quarterback" << endl;

cout << "2. Wide Receiver" << endl;

cout << "3. Tight End" << endl;

cout << "4. Running Back" << endl;

cout << "5. Offensive Lineman" << endl;

positionChoice = getMenuChoice(1,5);

*// Subtract 1 to get accurate enum static\_cast*

positionChoice--;

offensiveRole position = **static\_cast**<offensiveRole>(positionChoice);

Offense newOffensePlayer(playerName, playerID, position);

*// Adds a new unique\_ptr to team of players*

team.push\_back(make\_unique<Offense>(newOffensePlayer));

}

*// Makes new defense player*

**if**(userChoice == 2){

cout << endl << "What position does your defensive player play?" << endl;

cout << "1. Defensive Lineman" << endl;

cout << "2. Linebacker" << endl;

cout << "3. Defensive Back" << endl;

positionChoice = getMenuChoice(1,3);

*// Subtract 1 to get accurate enum static\_cast*

positionChoice--;

defensiveRole position = **static\_cast**<defensiveRole>(positionChoice);

Defense newDefensePlayer(playerName, playerID, position);

*// Adds new unique\_ptr to team of players*

team.push\_back(make\_unique<Defense>(newDefensePlayer));

}

*// Makes new special player*

**if**(userChoice == 3){

cout << endl << "What position does your special player play?" << endl;

cout << "1. Kicker" << endl;

cout << "2. Holder" << endl;

cout << "3. Punter" << endl;

cout << "4. Returner" << endl;

positionChoice = getMenuChoice(1,4);

*// Subtract 1 to get accurate enum static\_cast*

positionChoice--;

specialRole position = **static\_cast**<specialRole>(positionChoice);

specialPlayer newSpecialPlayer(playerName, playerID, position);

*// Adds new unique\_ptr to team of players*

team.push\_back(make\_unique<specialPlayer>(newSpecialPlayer));

}

}

**return** 0;

}

*// Returns a player ID*

int getPlayerID(){

int playerID;

cout << endl << "What is your player's number?" << endl;

bool valid = false;

**while**(!valid){

**try**{

cin >> playerID;

**if**(playerID < 0) **throw**(runtime\_error("Player ID cannot be less than 0."));

valid = true;

}**catch**(**const** exception& e){

cerr << endl << e.what() << endl;

cin.clear();

cin.ignore(10000, '\n');

}

}

**return** playerID;

}

*// Gets a generic menu choice between lower bound and upper bound*

int getMenuChoice(int lowerBound, int upperBound){

int userChoice;

bool valid = false;

**while**(!valid){

**try**{

cin >> userChoice;

**if**(userChoice < lowerBound || userChoice > upperBound)

**throw**(runtime\_error("Please choose an option between " + to\_string(lowerBound) + " and " + to\_string(upperBound) + "."));

valid = true;

}**catch**(**const** exception& e){

cerr << endl << e.what() << endl;

cin.clear();

cin.ignore(10000, '\n');

}

}

**return** userChoice;

}

# 5 Three Use Case Screen Shots

















