BrightNetwork Technology Internship
Experience UK

Software Development – **C++**

Morgan Ruffell

UCA – BSc Games Technology

Contents

Platform Details

- Microsoft Visual Studio 2019
- Task Selected: C++
- C++14
- Installed Plugins
  - Visual Assist

Additional Class Modifications

Solution required extension to pre-existing class structures

Video Class

```
/**
 * A class used to represent a video.
 */
class Video
{
private:
    std::string mTitle;
    std::string mVideoId;
    std::vector<std::string> mTags;

public:
    Video(std::string&& title, std::string&& videoId,
        std::vector<std::string>&& tags);

    std::string TagReason;

    static bool IsFlagged;

    // Returns the title of the video.
    const std::string& getTitle() const;

    // Returns the video id of the video.
    const std::string& getVideoId() const;

    // Returns a readonly collection of the tags of the video.
    const std::vector<std::string>& getTags() const;
};
```

```
1   #include "video.h"
2
3   #include <iostream>
4   #include <utility>
5   #include <vector>
6
7   bool Video::IsFlagged = false;
8
9   Video::Video(std::string&& title, std::string&& videoId,
10      ............std::vector<std::string>&& tags) :
11      mTitle(std::move(title)),
12      mVideoId(std::move(videoId)),
13      mTags(std::move(tags)) {
14  }
15
16  const std::string& Video::getTitle() const { return mTitle; }
17
18  const std::string& Video::getVideoId() const { return mVideoId; }
19
20  const std::vector<std::string>& Video::getTags() const { return mTags; }
21
```

- Created a static bool for the flagged nature of an individual video object

Video Player Class

- New public section for helper methods
- Stores the currently playing video as a string
- Playlists are a vector of vectors of strings

```
class VideoPlayer {
private:

public:
    VideoLibrary mVideoLibrary;
    VideoPlayer() = default;

    // This class is not copyable to avoid expensive copies.
    VideoPlayer(const VideoPlayer&) = delete;
    VideoPlayer& operator=(const VideoPlayer&) = delete;

    // This class is movable.
    VideoPlayer(VideoPlayer&&) = default;
    VideoPlayer& operator=(VideoPlayer&&) = default;

public:

    //The video that is currently playing
    std::string CurrentlyPlayingVideo = "";
    bool IsVideoPaused;

    std::vector<std::vector<std::string>> Playlists;


    void numberOfVideos();
    void showAllVideos();
    void playVideo(const std::string& videoId);
    void stopVideo();
    void playRandomVideo();
    void pauseVideo();
    void continueVideo();
    void showPlaying();
    void createPlaylist(const std::string& playlistName);
    void addVideoToPlaylist(const std::string& playlistName, const std::string& videoId);
    void showAllPlaylists();
    void showPlaylist(const std::string& playlistName);
    void removeFromPlaylist(const std::string& playlistName, const std::string& videoId);
    void clearPlaylist(const std::string& playlistName);
    void deletePlaylist(const std::string& playlistName);
    void searchVideos(const std::string& searchTerm);
    void searchVideosWithTag(const std::string& videoTag);
    void flagVideo(const std::string& videoId);
    void flagVideo(const std::string& videoId, const std::string& reason);
    void allowVideo(const std::string& videoId);


//These are methods I implemented as part of my solution
public:

    bool DoesVideoExist(std::string videoId);
    bool AreAllVideosFlagged();
    bool IsVideoFlagged(std::string VideoName);

    void AddVideo(const std::string& playlistName, const std::string& videoId);
    bool IsVideoAlreadyInCollection(const std::string& playlistName, const std::string& videoId);
    void CleanOutPlaylist(const std::string& playlistName, bool IsPlaylistCleared);
```

Additional Methods

I created a few additional methods in my solution to modularise code that was inside of the stubbed out methods.

```
//These are methods I implemented as part of my solution
public:

    bool DoesVideoExist(std::string videoId);
    bool AreAllVideosFlagged();
    bool IsVideoFlagged(std::string VideoName);

    void AddVideo(const std::string& playlistName, const std::string& videoId);
    bool IsVideoAlreadyInCollection(const std::string& playlistName, const std::string& videoId);
    void CleanOutPlaylist(const std::string& playlistName, bool IsPlaylistCleared);
};
```

- These are the declaration of some helper methods that I implemented as part of my solution.
- The definitions are available inside of the Methods and Solution section of this documentation.

Methods & Solution

Final Source Code

Header & Implementation code included

ShowAll & Exist

- Solution to the show all videos and the does video exist
- For every video in the instance of the video library, we get the title of the video and print it out along with the video id.
- As well as this we also go through all of the tags of the videos
- For the does video exist we iterate through the whole collection of the videos that are in that instance of the video library
- If that at the index of I, we get the id at the element and if that id is equal to the string passed in then we return true.
- The Does video exist is a helper method I created to help with my solution.

```
void VideoPlayer::numberOfVideos()
{
    std::cout << mVideoLibrary.getVideos().size() << " videos in the library" << std::endl;
}

void VideoPlayer::showAllVideos()
{
    for (auto x : mVideoLibrary.getVideos())
    {
        std::cout << x.getTitle() << " (" << x.getVideoId() << ") ";
        std::cout << "[";

        for (auto TagIndex : x.getTags())
        {
            std::cout << TagIndex << " ";
        }

        std::cout << "]" << std::endl;
    }
}

bool VideoPlayer::DoesVideoExist(std::string videoId)
{
    for (int i = 0; i < mVideoLibrary.getVideos().size();)
    {
        if (mVideoLibrary.getVideos().at(i).getVideoId() == videoId)
        {
            return true;
        }

        i++;
    }

    return false;
}
```

Play Video

- Solution to the play video, includes instances of helper methods I made to help determine existence and if it's flagged or not.
- Given more time I would refactor this into more modular code design so that there is more function calls within this block. It's not very readable.

```
bool VideoPlayer::IsVideoFlagged(std::string VideoTag)
{
    int VideoCount = mVideoLibrary.getVideos().size();
    int Count = 0;

    for (auto Ivideo : mVideoLibrary.getVideos())
    {
        if (Ivideo.IsFlagged == true && mVideoLibrary.getVideo(VideoTag)->getVideoId() == VideoTag)
        {
            return true;
        }
    }

    return false;
}
```

```
void VideoPlayer::playVideo(const std::string& videoId)
{
    if (IsVideoFlagged(videoId) == false)
    {
        if (DoesVideoExist(videoId))
        {
            //Get a specific video - account for paused too?
            if (IsVideoPaused)
            {
                std::cout << "The Currently playing video is paused" << std::endl;
                std::cout << "Playing" << CurrentlyPlayingVideo << std::endl;
            }

            else
            {
                if (!CurrentlyPlayingVideo.empty())
                {
                    CurrentlyPlayingVideo = mVideoLibrary.getVideo(videoId)->getTitle();
                }

                else
                {
                    char condition;

                    std::cout << "There is currently a video playing, would you like to stop it?" << std::endl;
                    std::cout << "y - for yes, n - for no" << std::endl;

                    std::cin >> &condition;

                    if (condition == 'y' || condition == 'Y')
                    {
                        CurrentlyPlayingVideo = mVideoLibrary.getVideo(videoId)->getTitle();
                        return;
                    }

                    if (condition == 'n' || condition == 'N')
                    {
                        std::cout << "Continuing to play " << CurrentlyPlayingVideo;
                        return;
                    }
                }
            }
        }

    }

    else
    {
        std::cout << "Cannot play video, id is invalid. It does not exist within the library!" << std::endl;
    }
}
```

Stop Video

- Stop Video method, this also checks for user input using a design pattern that is commonplace throughout this documentation.
- This also contains Is video paused as well.

```
void VideoPlayer::stopVideo()
{
    if (IsVideoPaused)
    {
        std::cout << "The Currently playing video is paused" << std::endl;
        std::cout << "Stopping" << CurrentlyPlayingVideo << std::endl;
    }

    if (CurrentlyPlayingVideo.empty())
    {
        std::cout << "Cannot stop video: No video is currently playing" << std::endl;
    }

    else
    {
        std::cout << "There is currently" << CurrentlyPlayingVideo << " playing." << std::endl;
        std::cout << "Do you want to stop it?" << std::endl;

        std::cout << "y for yes" << std::endl;
        std::cout << "n for no" << std::endl;

        char condition;

        std::cin >> &condition;

        if (condition == 'y' || condition == 'Y')
        {
            CurrentlyPlayingVideo.clear();
            std::cout << "Stopping currently playing video";
        }
        else if (condition == 'n' || condition == 'N')
        {
            std::cout << "Continuing to play " << CurrentlyPlayingVideo;
        }
    }
}
```

Play Random Video

- Stop Video method, this also checks for user input using a design pattern that is commonplace throughout this documentation.
- This also contains Is video paused as well.

```
bool VideoPlayer::AreAllVideosFlagged()
{
    int VideoCount = mVideoLibrary.getVideos().size();
    int Count = 0;

    for (auto x : mVideoLibrary.getVideos())
    {
        if (x.IsFlagged == true)
        {
            Count + 1;

            if (Count == VideoCount)
            {
                return true;
            }
        }
    }

    return false;
}
```

```
void VideoPlayer::playRandomVideo()
{
    if (!AreAllVideosFlagged())
    {
        if (!CurrentlyPlayingVideo.empty())
        {
            std::cout << "Currently playing video is not empty. Would you like to stop the current video and continue?" << std::endl;

            char condition;

            std::cout << "y for yes" << std::endl;
            std::cout << "n for no" << std::endl;

            std::cin >> &condition;

            if (condition == 'y' || condition == 'Y')
            {
                CurrentlyPlayingVideo.clear();
                std::cout << "Stopping currently playing video";
            }
            else if (condition == 'n' || condition == 'N')
            {
                std::cout << "Continuing to play " << CurrentlyPlayingVideo;
                return;
            }

        }

        int MaxBound = mVideoLibrary.getVideos().size();
        int MinBound = 0;
        int RandomNumber = rand() % MaxBound;

        std::cout << "Playing Random Video..." << std::endl;

        CurrentlyPlayingVideo = mVideoLibrary.getVideos().at(RandomNumber).getTitle();
        std::cout << "Random video selected is " << CurrentlyPlayingVideo << std::endl;
    }

    else
    {
        std::cout << "No videos available" << std::endl;
    }
}
```

Pause & Continue Video

- This is the paused and continue video methods.
- This also makes use of the Video paused Boolean that lives on the video player class.

```
void VideoPlayer::pauseVideo()
{
    if (IsVideoPaused)
    {
        std::cout << "Pausing video: " << CurrentlyPlayingVideo << std::endl;
    }
    else
    {
        IsVideoPaused = true;
        std::cout << "Video already paused: " << CurrentlyPlayingVideo << std::endl;
    }
}

void VideoPlayer::continueVideo()
{
    if (!CurrentlyPlayingVideo.empty())
    {
        if (IsVideoPaused)
        {
            !IsVideoPaused;
            std::cout << CurrentlyPlayingVideo << " Is now playing!" << std::endl;
        }
        else
        {
            std::cout << CurrentlyPlayingVideo << " Is currently playing." << std::endl;
        }
    }
    else
    {
        std::cout << "Cannot continue video: No video is currently playing";
    }
}
```

Show Playing & Create Playlist

- This is the show playing and create playlist methods.
- Playlists are stored in a std::vector<std::vector<std::string>> a vector of a vectors of strings.

```
void VideoPlayer::showPlaying()
{
    if (!CurrentlyPlayingVideo.empty())
    {
        std::cout << "The currently playing video is " << CurrentlyPlayingVideo << std::endl;

        if (IsVideoPaused)
        {
            std::cout << CurrentlyPlayingVideo << "Is currently paused.";
        }
        else
        {
            std::cout << CurrentlyPlayingVideo << "Is playing! Enjoy!";
        }
    }

    else
    {
        std::cout << "No video is currently playing" << std::endl;
    }

}

void VideoPlayer::createPlaylist(const std::string& playlistName)
{
    //The first element of a playlist is the name --- questionable but can't think of a better solution rn
    std::vector<std::string> Temp;

    Temp.push_back(playlistName);
    Playlists.push_back(Temp);

    std::cout << "Successfully created new playlist" << playlistName << "Currently contains no videos, add some through 'Add_Video_To_Playlist'" << std::endl;
}
```

Add video to playlist

- This checks to see if the video is already in collection, this had the same parameters as the method it is nested within.
- Now inside of the Add video method we use ranged for loops to iterate through every playlist in playlists, pushing back the video based on video id.

```
void VideoPlayer::addVideoToPlaylist(const std::string& playlistName, const std::string& videoId)
{
    if (IsVideoAlreadyInCollection(playlistName, videoId) == false)
    {
        if (DoesVideoExist(videoId))
        {
            AddVideo(playlistName, videoId);
            std::cout << "Added video to " << playlistName << mVideoLibrary.getVideo(videoId)->getTitle() << std::endl;
        }
        else
        {
            std::cout << "Cannot add video to " << playlistName << ": Video does not exist";
        }
    }
    else
    {
        std::cout << " Cannot add video to " << playlistName << " : video already added";
    }
}

void VideoPlayer::AddVideo(const std::string& playlistName, const std::string& videoId)
{
    //for each playlist in playlists - This is a vector of a vector of strings.
    for (auto Playlist : Playlists)
    {
        //This should allow us to access element at element 0 of the playlist - It's name.
        if (Playlist.at(0) == playlistName)
        {
            Playlist.push_back(mVideoLibrary.getVideo(videoId)->getVideoId());
        }
    }
}
```

Checking Collections & Show all Playlists

- The code comments are pretty explanatory as to what is going on here
- I'm trying to keep most of these algorithms O(n), but because there are nested loops. These will become quadratic quite easily. If I'm getting that wrong my apologies, I'm not used to time and space complexity.

```
bool VideoPlayer::IsVideoAlreadyInCollection(const std::string& playlistName, const std::string& videoId)
{
    //for each playlist in playlists -- This is a vector of a vector of strings.
    for (auto Playlist : Playlists)
    {
        //This should allow us to access element at element 0 of the playlist -- It's name.
        if (Playlist.at(0) == playlistName)
        {
            //This is another quadratic algorithm, because of the small size of the data set, this is fine.
            //Even in the worst case it's fine. If this was larger data set I'd use a more computationally cheap algorithm.
            for (auto Video : Playlist)
            {
                if (Video == mVideoLibrary.getVideo(videoId)->getTitle())
                {
                    return true;
                }

                return false;
            }
        }
    }
}

void VideoPlayer::showAllPlaylists()
{
    bool AnyPlaylists = false;

    if (Playlists.empty())
    {
        AnyPlaylists = false;
    }

    if (AnyPlaylists == true)
    {
        std::cout << "Showing all playlists:" << std::endl;

        for (auto playlist : Playlists)
        {
            for (auto video : playlist)
            {
                if (video.at(0))
                {
                    std::cout << "  " << video.at(0);
                }
            }
        }
    }
    else
    {
        std::cout << "No playlists exist yet" << std::endl;
    }

}
```

Show Playlist & Remove From

- Theses are the methods used to show all of the contents of a specific playlist, as well as removing videos from playlists.
- I'm trying to utilise clean coding practices throughout my implementation. Ensuring that variable names make more sense than just a or b.

```
void VideoPlayer::showPlaylist(const std::string& playlistName)
{
    for (auto playlist : Playlists)
    {
        if (playlist.at(0) == playlistName)
        {
            for (auto video : playlist)
            {
                std::cout << video << std::endl;
            }
        }
    }

    std::cout << "showPlaylist needs implementation" << std::endl;
}

void VideoPlayer::removeFromPlaylist(const std::string& playlistName, const std::string& videoId)
{
    for (auto playlist : Playlists)
    {
        if (playlist.at(0) == playlistName)
        {
            for (auto video : playlist)
            {
                if (video == mVideoLibrary.getVideo(videoId)->getVideoId())
                {
                    video.erase();
                    std::cout << "Removed video from " << playlistName << ":" << mVideoLibrary.getVideo(videoId)->getTitle() << std::endl;
                    return;
                }
            }
        }

    }

    std::cout << "Cannot remove video from " << playlistName << ": Playlist does not exist";
    return;
}
```

Clear Playlist

- In this we can clear playlists. If I had more time I would modularise a lot of this code, as there are a lot of checks and loops inside of this.
- Clean out playlist is another one that I've made to help in this. In this it just removes a lot of the logic from the main clearPlaylist and makes the code easier to read.

```
void VideoPlayer::clearPlaylist(const std::string& playlistName)
{
    bool isPlaylistCleared = false;
    bool DoesPlaylistExist = false;

    if (isPlaylistCleared == false)
    {
        if (DoesPlaylistExist == false)
        {
            for (auto playlist : Playlists)
            {
                if (playlist.at(0) == playlistName)
                {
                    DoesPlaylistExist = true;
                }

                if (DoesPlaylistExist == true)
                {
                    CleanOutPlaylist(playlistName, isPlaylistCleared);

                    if (isPlaylistCleared == true)
                    {
                        std::cout << "Successfully removed all videos from " << playlistName << std::endl;
                        return;
                    }
                }
            }
        }
    }
}
```

```
void VideoPlayer::CleanOutPlaylist(const std::string& playlistName, bool IsPlaylistCleared)
{
    for (auto playlist : Playlists)
    {
        if (playlist.at(0) == playlistName)
        {
            for (auto video : playlist)
            {
                if (video != playlistName)
                {
                    video.clear();
                }
            }
        }

    }

    IsPlaylistCleared = true;
}
```

Deleting & Search Playlist

- The delete and search playlist method implementations.
- Again I'm new to big O notation and understanding time and space complexity, forgive me if I'm wrong!
- I'm just using simple search algorithms because of the fact that this is a small data set.

```
void VideoPlayer::deletePlaylist(const std::string& playlistName)
{
    for (auto playlist : Playlists)
    {
        for (auto video : playlist)
        {
            //I made a bad choice design using the name of the playlist as the first element. It won't play from it but...
            if (video == playlistName)
            {
                //clearing all the elements in the vector -- I was thinking of using new and delete to do dynamic
                //memory allocation so that the memory where it is stored would be freed, but it's a bit scary with an object of this size.
                playlist.clear();
                std::cout << "Deleted playlist: " << playlistName << std::endl;
            }
        }
    }

    std::cout << "Cannot delete playlist " << playlistName << "Playlist does not exist";
}

void VideoPlayer::searchVideos(const std::string& searchTerm)
{
    //Simple linear search, this is a small data set. If it was larger a different sorting alogrithim would be used.
    //In the case of something where time & space complexity is important I would use a merge sort. O(n) + O(1)
    for (auto x : mVideoLibrary.getVideos())
    {
        if (x.getTitle() == searchTerm)
        {
            std::cout << x.getTitle() << std::endl;
        }
    }
}

void VideoPlayer::searchVideosWithTag(const std::string& videoTag)
{
    std::vector<std::string> VideoTag;
    VideoTag.push_back(videoTag);

    //Quadratic O(n2?)
    for (int i = 0; i > mVideoLibrary.getVideos().size(); i++)
    {
        if (mVideoLibrary.getVideos().at(i).getTags() == VideoTag)
        {
            std::cout << mVideoLibrary.getVideos().at(i).getTitle() << std::endl;
        }
    }
}
```

Flagging Videos

- This and the following page are the implementations for flagging videos, as well as the implementation for the allow videos
- The same code for processing user input is also present inside of these methods as well.

```
void VideoPlayer::flagVideo(const std::string& videoId)
{
    //Pretty bad, but I'm not sure how to do this as well.
    if (mVideoLibrary.getVideo(videoId)->IsFlagged == true)
    {
        std::cout << mVideoLibrary.getVideo(videoId)->getTitle() << "Is Currently flagged by the user";
    }

    else
    {
        std::cout << mVideoLibrary.getVideo(videoId)->getTitle() << "Is not currently flagged by the user, would you like to flag it?";

        char condition;

        std::cout << "y for yes" << std::endl;
        std::cout << "n for no" << std::endl;

        std::cin >> &condition;

        if (condition == 'y' || condition == 'Y')
        {
            mVideoLibrary.getVideo(videoId)->IsFlagged = true;

            if (!CurrentlyPlayingVideo.empty())
            {
                stopVideo();
            }

            std::cout << "Video was flagged";
        }
        else if (condition == 'n' || condition == 'N')
        {
            std::cout << "Video was not flagged";
        }
    }
}
```

```
void VideoPlayer::flagVideo(const std::string& videoId, const std::string& reason)
{
    if (mVideoLibrary.getVideo(videoId)->IsFlagged == true)
    {
        std::cout << mVideoLibrary.getVideo(videoId)->getTitle() << "Is Currently flagged by the user";
        std::cout << "It is currently flagged " << mVideoLibrary.getVideo(videoId)->TagReason;
    }
    else
    {
        std::cout << mVideoLibrary.getVideo(videoId)->getTitle() << "Is not currently flagged by the user, would you like to flag it?";

        char condition;

        std::cout << "y for yes" << std::endl;
        std::cout << "n for no" << std::endl;

        std::cin >> &condition;

        if (condition == 'y' || condition == 'Y')
        {
            mVideoLibrary.getVideo(videoId)->TagReason + reason;
            mVideoLibrary.getVideo(videoId)->IsFlagged = true;

            std::cout << "Video was flagged" << " with the tag: " << reason;
        }
        else if (condition == 'n' || condition == 'N')
        {
            std::cout << "Video was not flagged";
        }
    }
}

void VideoPlayer::allowVideo(const std::string& videoId)
{
    std::string NullTag = "";

    if (mVideoLibrary.getVideo(videoId)->IsFlagged == true)
    {
        std::cout << mVideoLibrary.getVideo(videoId)->getTitle() << "Is Currently flagged by the user";
        std::cout << "It is currently flagged " << mVideoLibrary.getVideo(videoId)->TagReason;
        std::cout << "Would you like to unflag it?" << std::endl;

        char condition;

        std::cout << "y for yes" << std::endl;
        std::cout << "n for no" << std::endl;

        std::cin >> &condition;

        if (condition == 'y' || condition == 'Y')
        {
            mVideoLibrary.getVideo(videoId)->IsFlagged = false;
            std::cout << "Video was unflagged" << std::endl;
        }
        else if (condition == 'n' || condition == 'N')
        {
            std::cout << "Video was not unflagged";
        }
    }
}
```

External Links

- Visual Assist - https://www.wholetomato.com/
- Visual Studio 2019 Professional - https://visualstudio.microsoft.com/
- Compiler – MSVC
- GitHub Repository - https://github.com/MorganRuffell/BN-TechnologyExperience-SoftwareDevelopmentSolutionCpp