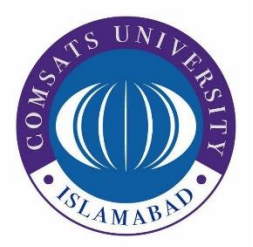


# COMSATS University Islamabad (CUI)

## Project Report

## for

## Music Player

Version 1.0

***By***

[redacted]

***Instructor***

[redacted]

**CSC321 – Microprocessor and Assembly Language**

***Bachelor of Science in Computer Science (2021-2025)***

# Table of Contents

**Project Category:**

☐ **A-**Desktop Application

# Abstract

The music player, developed using the Irvine32 library, offers a robust set of features including music playback controls, playlist management, and user interaction. Designed as both a functional tool for music enthusiasts and an educational resource, it provides a hands-on experience in memory management, API interactions, and GUI development within the challenging landscape of assembly language programming. The player showcases a seamless integration of these components, creating an immersive environment for users to explore and enjoy their music while gaining insights into the intricacies of low-level programming.

# 1. Introduction

The Music Player is a console-based interactive application developed in Irvine32 Assembly language. This music player exemplifies the fusion of low-level programming and multimedia functionality. Operating seamlessly on the x86 architecture, the player showcases the efficiency and compactness achievable through assembly language, providing enthusiasts and learners with a unique insight into the intricacies of program development at the lowest level. With music playback controls, playlist management, and user interaction, the project serves both as a functional tool for enthusiasts and an educational resource, providing hands-on experience in memory management, API interactions, and GUI development within assembly language programming.

# 2. Description

This Music Player, crafted using the Irvine32 library, presents a comprehensive set of features such as music playback controls and playlist management. Tailored for both music enthusiasts and learners, it offers a practical exploration of memory management, API interactions, and GUI development within the intricate realm of assembly language programming. With a seamless integration of these functionalities, users can enjoy a rich musical experience while delving into the intricacies of low-level programming. The player not only serves as a functional tool but also doubles as an educational resource, making it a versatile and engaging project for individuals interested in both music and assembly language programming.

# 3. Motivation

The reason for making this music player with the Irvine32 library is simply to have fun with music and learn cool computer stuff at the same time. Imagine creating your own music playground where you control the beats and melodies. By doing this project, we're not just making something to enjoy our favorite tunes; we're also discovering how computers handle memory, talk to other programs (APIs), and create those buttons and sliders we see on screens. It's like turning our love for music into a super cool computer adventure.

# 4. Procedures

## 4.1 InitializeAndRunDialog

- It initializes the application by getting the module handle, initializing common controls, and creating a dialog box from a template in the application's resources.
- It ends the process and all its threads after the dialog box is closed, with the exit code being the value returned by DialogBoxParam

```
WinMain PROC
    invoke  GetModuleHandle, NULL
    mov hInstance, eax
    invoke  InitCommonControls
    invoke  DialogBoxParam, hInstance, IDD_DIALOG, 0, offset WinProc, 0
    invoke  ExitProcess, eax
WinMain ENDP
```

## 4.2 InitializeUIAndLoadPlaylist

- It loads a playlist from a text file, sets a timer to refresh every 0.2 seconds, and loads and sets images for the Play, Recycle, and Sound buttons.
- It sets the range and initial position of the volume slider control.

```
init proc hWnd:DWORD
    ;LOCAL hFile: DWORD
    ;LOCAL bytes_read: DWORD

    invoke LoadPlayListFromTXT, hWnd

    invoke SetTimer, hWnd, 1, 500, NULL

    mov eax, ICO_START
    invoke LoadImage, hInstance, eax,IMAGE_ICON,32,32,NULL
    invoke SendDlgItemMessage,hWnd,IDC_PlayButton, BM_SETIMAGE, IMAGE_ICON, eax

    mov eax, ICO_PLAYRECYCLE
    invoke LoadImage, hInstance, eax,IMAGE_ICON,32,32,NULL
    invoke SendDlgItemMessage,hWnd,IDC_RecycleButton, BM_SETIMAGE, IMAGE_ICON, eax

    mov eax, ICO_SOUNDOPEN
    invoke LoadImage, hInstance, eax,IMAGE_ICON,32,32,NULL
    invoke SendDlgItemMessage,hWnd,IDC_SoundButton, BM_SETIMAGE, IMAGE_ICON, eax

    invoke SendDlgItemMessage, hWnd, IDC_VolumeSlider, TBM_SETRANGEMIN, 0, 0
    invoke SendDlgItemMessage, hWnd, IDC_VolumeSlider, TBM_SETRANGEMAX, 0, 1000
    invoke SendDlgItemMessage, hWnd, IDC_VolumeSlider, TBM_SETPOS, 1, 1000
    ret
init endp
```

## 4.3 EndDialogAndSavePlaylist

- It ends the dialog box associated with the given window handle (hWnd).
- It saves the current playlist to a text file.

```
end_proc proc hWnd:DWORD
      invoke   EndDialog, hWnd, 0
      invoke   SavePlayListToTXT, hWnd
      ret
end_proc endp
```

## 4.4 SwitchCycleModeAndUpdateIcon

- It changes the cycle_mode based on its current state. If it's SINGLE_CYCLE, it changes to LIST_CYCLE. If it's LIST_CYCLE, it changes to RANDOM_CYCLE. If it's RANDOM_CYCLE, it changes back to SINGLE_CYCLE.
- After changing the cycle_mode, it loads the corresponding icon and sets it as the image for the RecycleButton.

```
change_cycle proc hWnd: DWORD
    .if cycle_mode == SINGLE_CYCLE
        mov cycle_mode, LIST_CYCLE
        mov eax, ICO_PLAYRECYCLE
    .elseif cycle_mode == LIST_CYCLE
        mov cycle_mode, RANDOM_CYCLE
        mov eax, ICO_PLAYRANDOM
    .elseif cycle_mode == RANDOM_CYCLE
        mov cycle_mode, SINGLE_CYCLE
        mov eax, ICO_PLAYSINGLE
    .endif

    invoke LoadImage, hInstance, eax,IMAGE_ICON,32,32,NULL
    invoke SendDlgItemMessage,hWnd,IDC_RecycleButton, BM_SETIMAGE, IMAGE_ICON, eax;
    ret
change_cycle endp
```

## 4.5 ToggleSoundAndUpdateIcon

- It toggles the have_sound variable. If have_sound is 1, it sets it to 0 and vice versa. It then loads the corresponding icon (ICO_SOUNDCLOSE when sound is off, ICO_SOUNDOPEN when sound is on).
- It updates the SoundButton with the loaded icon and adjusts the volume based on the current state of have_sound.

```
change_silence proc hWnd:DWORD
    .if have_sound == 1
        mov have_sound,0
        mov eax, ICO_SOUNDCLOSE
    .else
        mov have_sound,1
        mov eax, ICO_SOUNDOPEN
    .endif

    invoke LoadImage, hInstance, eax,IMAGE_ICON,32,32,NULL
    invoke SendDlgItemMessage,hWnd,IDC_SoundButton, BM_SETIMAGE, IMAGE_ICON, eax;
    invoke alter_volume, hWnd;
    Ret
change_silence endp
```

## 4.6 AdjustVolumeBasedOnSoundState

- It retrieves the current position of the volume slider control and prepares a command to set the volume accordingly.
- If the sound is on (have_sound == 1), it sets the volume to the slider's position. If the sound is off, it sets the volume to 0. It then sends the command to the Media Control Interface (MCI).

```
alter_volume proc hWin: DWORD

    invoke SendDlgItemMessage,hWin,IDC_VolumeSlider,TBM_GETPOS,0,0
    .if have_sound == 1
        invoke wsprintf, addr mci_cmd, addr cmd_setVol,eax
    .else
        invoke wsprintf, addr mci_cmd, addr cmd_setVol,0
    .endif
    invoke mciSendString, addr mci_cmd, NULL, 0, NULL
    Ret
alter_volume endp
```

## 4.7 DisplayCurrentVolumeLevel

- It retrieves the current position of the volume slider control and calculates the volume level by dividing the position by 10.
- It then updates the VolumeText control to display the calculated volume level.

```
show_volume proc hWin: DWORD
    local temp: DWORD
    invoke SendDlgItemMessage,hWin,IDC_VolumeSlider,TBM_GETPOS,0,0
    mov temp, 10
    mov edx, 0
    div temp
    invoke wsprintf, addr mci_cmd, addr int_fmt, eax
    invoke SendDlgItemMessage, hWin, IDC_VolumeText, WM_SETTEXT,0,addr mci_cmd
    Ret
show_volume endp
```

## 4.8 TogglePlayPauseAndUpdateIcon

- It checks the current state of the music (music_state). If the music is stopped, it starts playing the music and changes the icon to pause. If the music is playing, it pauses the music and changes the icon to start. If the music is paused, it resumes playing the music and changes the icon to pause.
- After changing the music_state, it loads the corresponding icon and sets it as the image for the PlayButton.

```
handle_play proc hWnd:DWORD
    .if music_state == 0
        invoke OnPlayMusic
        invoke get_songlength, hWnd
        mov eax, ICO_PAUSE
    .elseif music_state == 1
        invoke OnPause
        mov eax, ICO_START
    .elseif music_state == 2
        invoke OnPlayMusic
        mov eax, ICO_PAUSE
    .endif

    invoke LoadImage, hInstance, eax,IMAGE_ICON,32,32,NULL
    invoke SendDlgItemMessage,hWnd,IDC_PlayButton, BM_SETIMAGE, IMAGE_ICON, eax
    ret
handle_play endp
```

## 4.9 RetrieveSongLengthAndUpdateSlider

- It retrieves the length of the current song using the mciSendString function with the cmd_getLen command, converts the length to an integer, and sets the maximum range of the TimeSlider control to this value.
- It then calculates the total minutes and seconds of the song by dividing the total length by scale_second and scale_minute respectively, and stores these values in total_minute and total_second.

```
get_songlength proc hWnd: DWORD
    invoke mciSendString, addr cmd_getLen, addr total, 32, NULL
    invoke StrToInt, addr total
    invoke SendDlgItemMessage, hWnd, IDC_TimeSlider, TBM_SETRANGEMAX, 0, eax
    invoke StrToInt, addr total
    mov edx, 0
    div scale_second

    mov edx, 0
    div scale_minute
    mov total_minute, eax
    mov total_second, edx
    ret
get_songlength endp
```

## 4.10 ChangeTrackPositionAndUpdatePlayState

- It retrieves the current position of the time slider control, prepares a command to set the position in the music track, and sends the command to the Media Control Interface (MCI).
- Depending on the current state of the music (music_state), it either pauses the music and changes the icon to start, or resumes the music and changes the icon to pause. It then updates the PlayButton with the loaded icon.

```
alter_time proc hWnd: DWORD
    invoke SendDlgItemMessage,hWnd,IDC_TimeSlider,TBM_GETPOS,0,0
    invoke wsprintf, addr mci_cmd, addr cmd_setPos, eax
    invoke mciSendString, addr mci_cmd, NULL, 0, NULL
    .if music_state == 1
        mov music_state, 0

        mov eax, ICO_START
        invoke LoadImage, hInstance, eax,IMAGE_ICON,32,32,NULL
        invoke SendDlgItemMessage,hWnd,IDC_PlayButton, BM_SETIMAGE, IMAGE_ICON, eax
        invoke mciSendString, addr mciBasePlayCmd, NULL, 0, NULL
```

```
    .elseif music_state == 2
        invoke mciSendString, addr mciBasePlayCmd, NULL, 0, NULL
        mov music_state, 1

        mov eax, ICO_START
        invoke LoadImage, hInstance, eax,IMAGE_ICON,32,32,NULL
        invoke SendDlgItemMessage,hWnd,IDC_PlayButton, BM_SETIMAGE, IMAGE_ICON, eax
    .endif
    ret
alter_time endp
```

## 4.11 HandleTrackEndAndSwitchTrack

- It checks if the current position in the track (position) is greater than or equal to the total length of the track (total). If it is, this means the track has finished playing.
- Depending on the current cycle_mode, it either replays the same track if the mode is SINGLE_CYCLE, moves to the next track if the mode is LIST_CYCLE, or moves to a random track if the mode is RANDOM_CYCLE.

```
switch_next proc hWin: DWORD
    local temp: DWORD
    invoke StrToInt, addr total
    mov temp, eax
    invoke StrToInt, addr position
    .if eax >= temp ;结束播放
        .if cycle_mode == SINGLE_CYCLE
            invoke OnPlayMusic
        .elseif cycle_mode == LIST_CYCLE
            invoke SendMessage, hWin, WM_COMMAND, IDC_NextSongImage, 0;
        .elseif cycle_mode == RANDOM_CYCLE
            invoke SendMessage, hWin, WM_COMMAND, IDC_PrevSongImage, 0
        .endif
    .endif
    Ret
switch_next endp
```

## 4.12 SwitchToNewSongAndUpdateLength

- If a song is currently playing (music_state != 0), it stops the song using the mciSendString function with the cmd_close command.
- It then updates the current_index to the newSongIndex, starts playing the new song using the OnPlayMusic function, and retrieves the length of the new song using the get_songlength function.

```
alter_song proc hWin:DWORD, newSongIndex: DWORD
    .if music_state != 0
        invoke mciSendString, ADDR cmd_close, NULL, 0, NULL
    .endif

    mov eax, newSongIndex
    mov current_index, eax
    invoke OnPlayMusic
    invoke get_songlength, hWin
    Ret
alter_song endp
```

## 4.13 StartPlayingSelectedSong

- If the music is not paused (music_state != 2), it stops the current song if it's playing (music_state == 4), loads the lyrics, gets the song at the current_index, prepares a command to play the song, and sends the command to the Media Control Interface (MCI).
- It then sets the music_state to 1 (playing) and sends a command to the MCI to start playing the song.

```
OnPlayMusic PROC
    ; mov al, 60
    ; movzx bl, Index
    ; mul bl
    ; mov esi, PlaylistOffset
    ; add esi, eax
    .if music_state != 2
        .if music_state == 4
            invoke mciSendString, ADDR mciClose, 0, 0, 0
        .endif
        invoke LoadLRC
        invoke GetSong, current_index
        invoke wsprintf, ADDR mciCmd, ADDR mciBasePlayCmd, ADDR thisSong._path
        ;invoke MessageBox, 0, ADDR mciCmd, 0, MB_OK
        invoke mciSendString, ADDR mciCmd, 0, 0, 0
    .endif

    mov music_state, 1
    invoke mciSendString, ADDR mciPlayCmd, 0, 0, 0
    ret
OnPlayMusic endp
```

## 4.14 Pause, previous and next Song

- OnPause: This procedure sends a command to pause the current song and sets the music_state to 2 (paused).

```
OnPause PROC
    ;invoke MessageBox, 0, ADDR mciCmd, 0, MB_OK
    invoke mciSendString, ADDR mciPauseCmd, NULL, 0, NULL
    mov music_state, 2
    ret
OnPause endp
```

- OnPrevSong: If the current song is not the first one in the playlist, this procedure decrements the current_index, sets the music_state to 4 (stopped), and starts playing the previous song.

```
OnPrevSong PROC
     .if current_index != 0
          mov music_state, 4
          dec current_index
          invoke OnPlayMusic
     .endif
     ret
OnPrevSong endp
```

- OnNextSong: If the current song is not the last one in the playlist, this procedure increments the current_index, sets the music_state to 4 (stopped), and starts playing the next song.

```
OnNextSong PROC USES eax
    mov eax, songMenuSize
    .if current_index != eax
         mov music_state, 4
         add current_index, 1
         invoke OnPlayMusic
    .endif
    ret
OnNextSong endp
```

## 4.15 Set and Get Song

- SetSong: This procedure sets the name and path of a song at a given index in the songMenu array.

```
SetSong PROC,
    index: DWORD,
    ptrSongName: PTR BYTE,
    ptrSongPath: PTR BYTE
    mov eax, index
    mov ebx, TYPE Song
    mul ebx
    mov edi, eax
    INVOKE lstrcpy, ADDR (Song PTR songMenu[edi])._name, ptrSongName
    INVOKE lstrcpy, ADDR (Song PTR songMenu[edi])._path, ptrSongPath
    ret
SetSong ENDP
```

- GetSong: This procedure retrieves the name and path of a song at a given index from the songMenu array and stores them in the thisSong structure.

```
GetSong PROC,
    index: DWORD
    mov eax, index
    mov ebx, TYPE Song
    mul ebx
    mov edi, eax
    INVOKE lstrcpy, ADDR thisSong._name, ADDR (Song PTR songMenu[edi])._name
    INVOKE lstrcpy, ADDR thisSong._path, ADDR (Song PTR songMenu[edi])._path
    ret
GetSong ENDP
```

## 4.16 LoadPlaylistFromFileAndUpdateUI

- It opens a text file containing a playlist, reads the size of the playlist and the playlist itself from the file. If there's an error in reading the file or the read size doesn't match the expected size, it sets the playlist size to 0.
- It then iterates over each song in the playlist, retrieves the song details, and adds the song name to the SongList control in the dialog box.

```
LoadPlayListFromTXT PROC,
    hWin: DWORD
    LOCAL hiFile: DWORD
    LOCAL bytesRead: DWORD

    INVOKE GetCurrentDirectory, SIZEOF songMenuFilePath - 20, ADDR songMenuFilePath
    invoke lstrcat, ADDR songMenuFilePath, ADDR songMenuFilename

    INVOKE CreateFile, ADDR songMenuFilePath, GENERIC_READ, 0, NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, 0
    mov hiFile, eax

    .IF hiFile == INVALID_HANDLE_VALUE
        mov songMenuSize, 0
        JMP RETURN
```

```
    .ELSE
        INVOKE ReadFile, hiFile, ADDR songMenuSize, SIZEOF songMenuSize, ADDR bytesRead, NULL
        .IF bytesRead != SIZEOF songMenuSize
            mov songMenuSize, 0
        .ELSE
            INVOKE ReadFile, hiFile, ADDR songMenu, SIZEOF songMenu, ADDR bytesRead, NULL
            .IF bytesRead != SIZEOF songMenu
                mov songMenuSize, 0
            .ENDIF
        .ENDIF
    .ENDIF

    mov ecx, 0
    .WHILE ecx < songMenuSize
        push ecx

        INVOKE GetSong, ecx
        INVOKE SendDlgItemMessage, hWin, IDC_SongList, LB_ADDSTRING, 0, ADDR thisSong._name

        pop ecx
        inc ecx
    .ENDW

RETURN:
    INVOKE CloseHandle, hiFile
    ret
LoadPlayListFromTXT ENDP
```

## 4.17 WritePlaylistToFile

- It creates a new file (or overwrites if it already exists) with the path specified in songMenuFilePath, and writes the size of the playlist and the playlist itself to the file.
- If there's an error in creating the file (if hiFile is INVALID_HANDLE_VALUE), it skips the writing steps and jumps to the end of the procedure.

```
SavePlayListToTXT PROC,
    hWin: DWORD
    LOCAL hiFile: HANDLE
    LOCAL bytesWritten: DWORD

    INVOKE CreateFile, ADDR songMenuFilePath, GENERIC_WRITE, 0, NULL, CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, 0
    mov hiFile, eax
    .IF hiFile == INVALID_HANDLE_VALUE
        ;INVOKE printf, ADDR str1
        ;INVOKE printf, ADDR ln
        JMP RETURN
    .ENDIF

    INVOKE WriteFile, hiFile, ADDR songMenuSize, SIZEOF songMenuSize, ADDR bytesWritten, NULL
    INVOKE WriteFile, hiFile, ADDR songMenu, SIZEOF songMenu, ADDR bytesWritten, NULL

    ;INVOKE printf, ADDR str2
    ;INVOKE printf, ADDR ln
RETURN:
    INVOKE CloseHandle, hiFile
    ret
SavePlayListToTXT ENDP
```

## 4.18 AddSongsFromDialogAndUpdateUI

- It opens a dialog box for the user to select one or more songs. For each selected song, it adds the song to the songMenu array and increments the songMenuSize and newSongCount.
- After all songs are added, it updates the SongList control in the dialog box with the names of the new songs and saves the updated playlist to a text file.

```
AddSongByDialog PROC USES eax ebx esi edi,
    hWin: DWORD
    LOCAL newSongCount: DWORD
    LOCAL index: DWORD
    mov newSongCount, 0

    mov al, 0
    mov edi, OFFSET openfilename
    mov ecx, SIZEOF openfilename
    cld
    rep stosb
    mov openfilename.lStructSize, SIZEOF openfilename
    mov eax, hWin
    mov openfilename.hwndOwner, eax
    mov eax, OFN_ALLOWMULTISELECT
    or eax, OFN_EXPLORER
    mov openfilename.Flags, eax
    mov openfilename.lpstrFilter, OFFSET szFilter
    mov openfilename.nMaxFile, nMaxFile
    mov openfilename.lpstrTitle, OFFSET szLoadTitle
    mov openfilename.lpstrInitialDir, OFFSET szInitDir
    mov openfilename.lpstrFile, OFFSET szOpenFileNames

    INVOKE GetOpenFileName, ADDR openfilename

    .IF eax == 1
        INVOKE lstrcpyn, ADDR szPath, ADDR szOpenFileNames, openfilename.nFileOffset
        INVOKE lstrcat, ADDR szPath, ADDR sep
```

```
    mov esi, OFFSET szOpenFileNames
    add si, openfilename.nFileOffset
    mov al, [esi]

    .WHILE al != 0
        mov szFileName, 0
        invoke lstrcat, ADDR szFileName, ADDR szPath
        invoke lstrcat, ADDR szFileName, esi

        INVOKE SetSong, songMenuSize, esi, ADDR szFileName
        add songMenuSize, 1
        add newSongCount, 1
```

```
            invoke lstrlen, esi
            inc eax
            add esi, eax

            mov al, [esi]
        .ENDW

        mov ecx, newSongCount
        L1:
            mov eax, songMenuSize
            sub eax, ecx
            mov index, eax

            ; ecx 有毒
            push ecx

            INVOKE GetSong, index
            INVOKE SendDlgItemMessage, hWin, IDC_SongList, LB_ADDSTRING, 0, ADDR thisSong._name

            pop ecx
            LOOP L1

    .ENDIF

    INVOKE SavePlayListToTXT, hWin

    ret
AddSongByDialog ENDP
```

## 4.19 RemoveSelectedSongAndUpdatePlaylist

- It deletes a song from the songMenu array and the SongList control in the dialog box based on the currently selected song in the SongList.
- It then shifts all songs after the deleted song up by one position in the songMenu array, decreases the songMenuSize by 1, and saves the updated playlist to a text file.

```
DeleteSong PROC,
    hWin: DWORD,
    LOCAL index: DWORD

    INVOKE SendDlgItemMessage, hWin, IDC_SongList, LB_GETCURSEL, 0, 0
    mov index, eax

    INVOKE SendDlgItemMessage, hWin, IDC_SongList, LB_DELETESTRING, eax, 0

    INVOKE GetSong, index

    ;INVOKE printf, ADDR thisSong._name
    ;INVOKE printf, ADDR ln
    ;INVOKE printf, ADDR thisSong._path
    ;INVOKE printf, ADDR ln
```

```
    mov ecx, index
    inc ecx
    .while ecx < songMenuSize
        push ecx

        mov eax, index
        inc eax

        INVOKE GetSong, eax
        INVOKE SetSong, index, ADDR thisSong._name, ADDR thisSong._path

        pop ecx
        inc ecx
        inc index
    .endw

    sub songMenuSize, 1

    INVOKE SetSong, songMenuSize, ADDR initSong._name, ADDR initSong._path

    INVOKE SavePlayListToTXT, hWin

    ret
DeleteSong ENDP
```

# 5. Working

The Music Player operates by loading audio files from the user's device or streaming them from the internet. The user, represented by a playlist, can select, play, pause, skip, or shuffle songs. The player dynamically adjusts the volume and quality level as the user changes the settings. Behind the scenes, the program utilizes functions for audio decoding, playback control, and metadata extraction. The application provides real-time feedback on song title, artist, album, duration, and progress. The use of multimedia libraries enables audio processing, creating a smooth and enjoyable listening experience.