Activity based

Project Report on

System Programming BTECCE22504

Project - I

Submitted to Vishwakarma University, Pune

Under the Initiative of

Contemporary Curriculum, Pedagogy, and Practice (C2P2)

By

ATHARVA SHEVATE

SRN No : 202201727

Roll No : 02

Div : E

Third Year Engineering

Department of Computer Engineering

Faculty of Science and Technology

Academic Year

2024-2025

i. Problem Statement

Symbol Resolution Analyzer

o Input: An object file main.o with unresolved symbols from a dynamically linked function expectedoutput
Unresolved Symbol: add (Referenced in main.o)

Resolved Symbol: add (Address: 0x00401000 from helper.o)

Importance of the Project

Symbol resolution is a critical step in linking during software compilation. When a program is split across multiple files, some functions or variables might not be fully defined in one object file but referenced. The process of resolving these references ensures that the final executable links all the functions and symbols correctly. Without proper symbol resolution, the program won't work, leading to errors or crashes. Tools like this analyzer assist developers in identifying and resolving these issues efficiently.

Algorithm

The basic workflow of the program follows these steps:

1. GUI Setup:
   - The program starts by creating a graphical user interface with text fields for input and output, and buttons to trigger different actions (Browse, Analyze, Copy, Save).
2. Browse for Object File:
   - The user can select an object file (e.g., main.o) using the "Browse" button. The selected file's path is displayed in an input text box.
3. Analyze Symbols:
   - When the user clicks the "Analyze Symbols" button, the program retrieves the file name and simulates symbol resolution. It randomly picks a symbol (e.g., add, subtract) and assigns it a memory address. This represents the resolution of an unresolved symbol to a defined address from a helper file (e.g., helper.o).
4. Display Output:
   - The resolved symbol and its corresponding address are displayed in the output text box.

5. Copy to Clipboard / Save to File:
    - The user can either copy the result to the clipboard or save it to a file using the respective buttons.
6. History Management:
    - The program maintains a history of up to 10 files that were analyzed, which is cleared when the program closes.

---

5. Summary of the Algorithm:

1. Start the program.
2. Set up the GUI with input fields, buttons, and output display.
3. On "Browse", select an object file.
4. On "Analyze", simulate resolving unresolved symbols by assigning them an address.
5. Display the result.
6. Allow the user to either copy the result to the clipboard or save it to a file.
7. Maintain a history of the last 10 analyzed files.
8. On exit, clear the history and close the application.

Code Structure

The code is structured into the following components:

- Header Inclusions: Libraries like windows.h, commdlg.h, and stdio.h are used for GUI, file operations, and standard I/O functionality.
- Global Variables: GUI components like text fields and buttons are defined globally for easy access throughout the application.
- Core Functions:
    - ClearHistory(): Frees memory for previously analyzed files.
    - ResolveSymbols(): Simulates symbol resolution by assigning an unresolved symbol a resolved memory address.
    - CopyToClipboard(): Copies the resolved symbols output to the system clipboard.

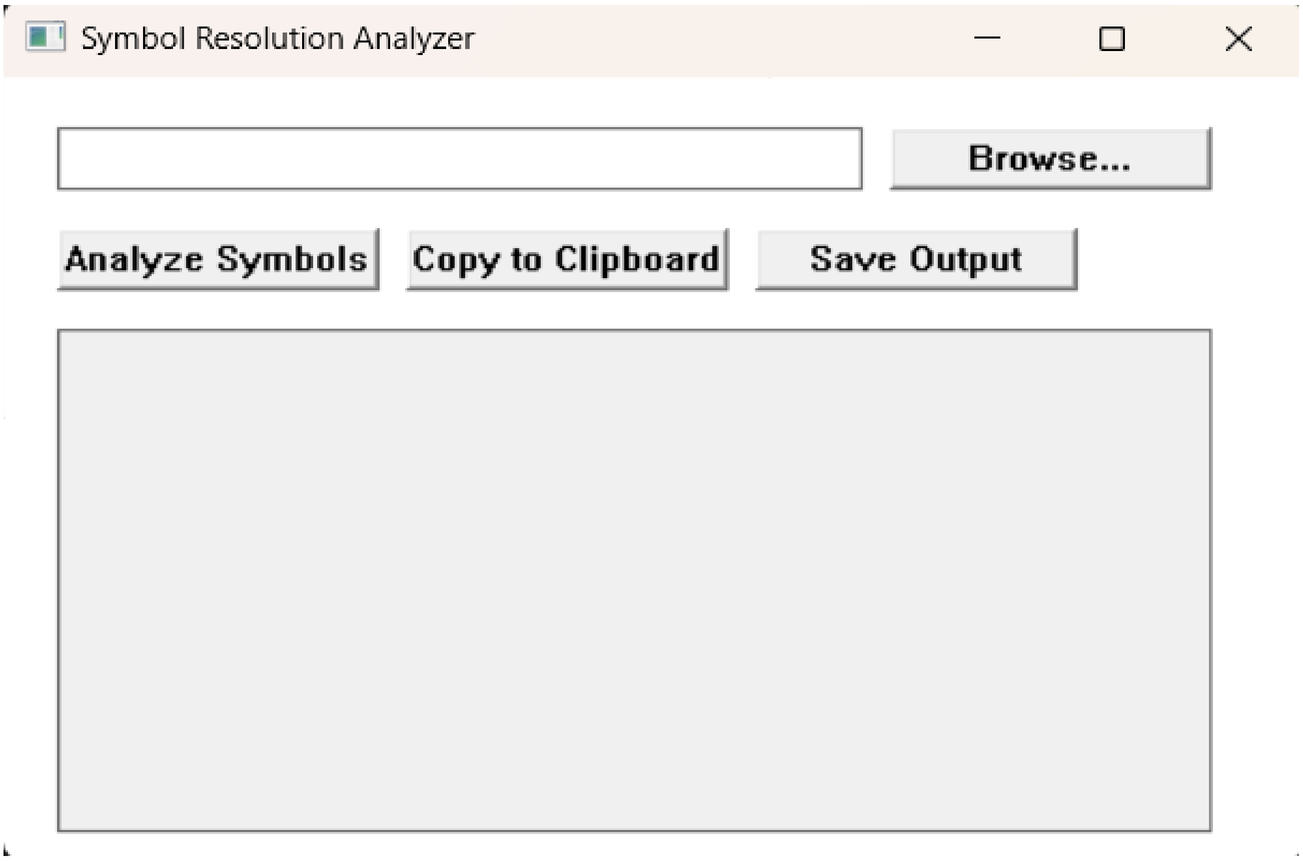
  - SaveOutputToFile(): Allows users to save the symbol resolution report to a file.
- WindowProc() Callback Function: Handles user interactions and actions on the GUI, such as file browsing, symbol analysis, copying, and saving results.
- Main Entry Point: Initializes the GUI and runs the message loop for the Windows application.

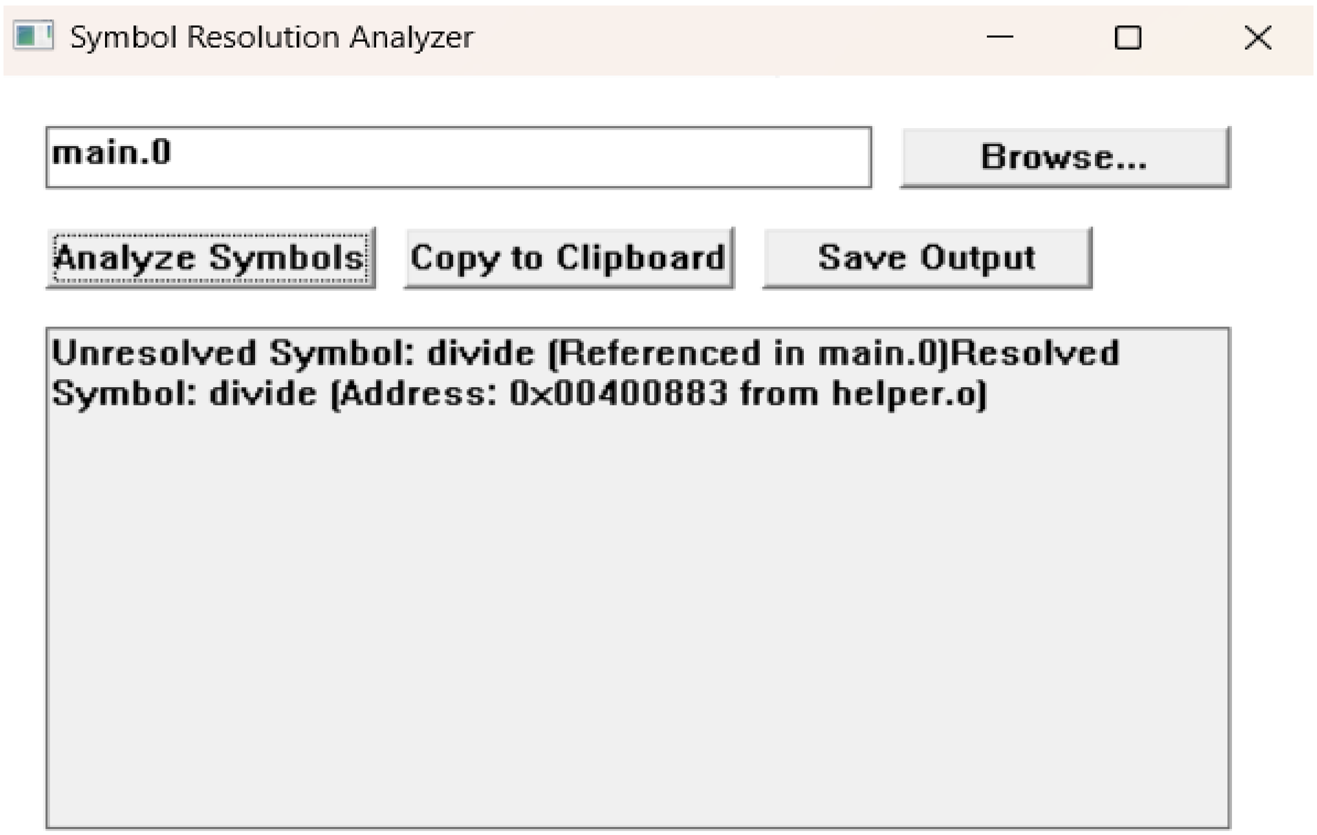
Code with Input and Output Screenshots

```
#include <windows.h>
#include <commdlg.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define MAX_HISTORY 10

// Global variables for GUI elements
HWND hEditInput, hEditOutput, hButtonAnalyze, hButtonBrowse, hButtonCopy,
hButtonSave;
char *history[MAX_HISTORY];
int historyIndex = 0;

// Function to clear the history
void ClearHistory() {
   for (int i = 0; i < MAX_HISTORY; i++) {
      free(history[i]);
   }
}

// Function to simulate symbol resolution
void ResolveSymbols(const char* inputFile, char* outputBuffer) {
    const char* symbols[] = {"add", "subtract", "multiply", "divide"};
    int symbolIndex = rand() % (sizeof(symbols) / sizeof(symbols[0]));
    unsigned int address = 0x00400000 + (rand() % 0x1000);

    sprintf(outputBuffer, "Unresolved Symbol: %s (Referenced in %s)\nResolved Symbol: %s
(Address: 0x%08X from helper.o)\n\n",
            symbols[symbolIndex], inputFile, symbols[symbolIndex], address);
}

// Function to copy text to clipboard
void CopyToClipboard(HWND hwnd, const char* text) {
    if (OpenClipboard(hwnd)) {
        EmptyClipboard();
        HGLOBAL hGlob = GlobalAlloc(GMEM_MOVEABLE, strlen(text) + 1);
        memcpy(GlobalLock(hGlob), text, strlen(text) + 1);
        GlobalUnlock(hGlob);
        SetClipboardData(CF_TEXT, hGlob);
        CloseClipboard();
    }
}

// Function to save output to a file
void SaveOutputToFile(HWND hwnd, const char* output) {
    OPENFILENAME ofn;
    char szFile[260];
    ZeroMemory(&ofn, sizeof(ofn));
    ofn.lStructSize = sizeof(ofn);
    ofn.hwndOwner = hwnd;
    ofn.lpstrFile = szFile;
    ofn.lpstrFile[0] = '\0';
    ofn.nMaxFile = sizeof(szFile);
    ofn.lpstrFilter = "Text Files (.txt)\0.txt\0All Files (.)\0*.*\0";
    ofn.nFilterIndex = 1;
    ofn.lpstrFileTitle = NULL;
    ofn.nMaxFileTitle = 0;
    ofn.lpstrInitialDir = NULL;
    ofn.Flags = OFN_OVERWRITEPROMPT;

    if (GetSaveFileName(&ofn)) {
        FILE *file = fopen(ofn.lpstrFile, "w");
        if (file) {
            fputs(output, file);
            fclose(file);
        }
    }
}

// Callback function to handle messages
LRESULT CALLBACK WindowProc(HWND hwnd, UINT uMsg, WPARAM wParam,
LPARAM lParam) {
    char inputFile[260], outputBuffer[2048];

    switch (uMsg) {
        case WM_CREATE:
            hEditInput = CreateWindowEx(0, "EDIT", "", WS_CHILD | WS_VISIBLE |
WS_BORDER,
                                        20, 20, 300, 25, hwnd, NULL, NULL, NULL);

            hButtonBrowse = CreateWindowEx(0, "BUTTON", "Browse...", WS_CHILD |
WS_VISIBLE,
                                          330, 20, 120, 25, hwnd, (HMENU) 1, NULL, NULL);

            hButtonAnalyze = CreateWindowEx(0, "BUTTON", "Analyze Symbols", WS_CHILD
| WS_VISIBLE,
                                          20, 60, 120, 25, hwnd, (HMENU) 2, NULL, NULL);

            hButtonCopy = CreateWindowEx(0, "BUTTON", "Copy to Clipboard", WS_CHILD |
WS_VISIBLE,
                              150, 60, 120, 25, hwnd, (HMENU) 3, NULL, NULL);

      hButtonSave = CreateWindowEx(0, "BUTTON", "Save Output", WS_CHILD |
WS_VISIBLE,
                              280, 60, 120, 25, hwnd, (HMENU) 4, NULL, NULL);

      hEditOutput = CreateWindowEx(0, "EDIT", "", WS_CHILD | WS_VISIBLE |
WS_BORDER | ES_MULTILINE | ES_AUTOVSCROLL | ES_READONLY,
                              20, 100, 430, 200, hwnd, NULL, NULL, NULL);
      break;

    case WM_COMMAND:
      if (LOWORD(wParam) == 1) {  // Browse button clicked
        OPENFILENAME ofn;
        char szFile[260];
        ZeroMemory(&ofn, sizeof(ofn));
        ofn.lStructSize = sizeof(ofn);
        ofn.hwndOwner = hwnd;
        ofn.lpstrFile = szFile;
        ofn.lpstrFile[0] = '\0';
        ofn.nMaxFile = sizeof(szFile);
        ofn.lpstrFilter = "Object Files (.o)\0.o\0All Files (.)\0*.*\0";
        ofn.nFilterIndex = 1;
        ofn.Flags = OFN_PATHMUSTEXIST | OFN_FILEMUSTEXIST;

        if (GetOpenFileName(&ofn)) {
          SetWindowText(hEditInput, ofn.lpstrFile);
        }
      }
      else if (LOWORD(wParam) == 2) {  // Analyze button clicked
        GetWindowText(hEditInput, inputFile, sizeof(inputFile));
        ResolveSymbols(inputFile, outputBuffer);
        SetWindowText(hEditOutput, outputBuffer);
```

```
            // Add to history
            if (historyIndex < MAX_HISTORY) {
                history[historyIndex] = strdup(inputFile);
                historyIndex++;
            }
        }
        else if (LOWORD(wParam) == 3) {  // Copy button clicked
            GetWindowText(hEditOutput, outputBuffer, sizeof(outputBuffer));
            CopyToClipboard(hwnd, outputBuffer);
        }
        else if (LOWORD(wParam) == 4) {  // Save button clicked
            GetWindowText(hEditOutput, outputBuffer, sizeof(outputBuffer));
            SaveOutputToFile(hwnd, outputBuffer);
        }
        break;

    case WM_DESTROY:
        ClearHistory();
        PostQuitMessage(0);
        break;

    default:
        return DefWindowProc(hwnd, uMsg, wParam, lParam);
    }
    return 0;
}

// Main function: Entry point for Windows application
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow) {
    srand((unsigned int)time(NULL));

    WNDCLASS wc = {0};
    wc.lpfnWndProc = WindowProc;
    wc.hInstance = hInstance;
    wc.lpszClassName = "SymbolResolutionAnalyzer";

    RegisterClass(&wc);

    HWND hwnd = CreateWindowEx(0, wc.lpszClassName, "Symbol Resolution Analyzer",
WS_OVERLAPPEDWINDOW,
                    CW_USEDEFAULT, CW_USEDEFAULT, 500, 350, NULL, NULL,
hInstance, NULL);

    if (hwnd == NULL) {
        return 0;
    }

    ShowWindow(hwnd, nCmdShow);

    MSG msg = {0};
    while (GetMessage(&msg, NULL, 0, 0)) {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }

    return 0;
}
```

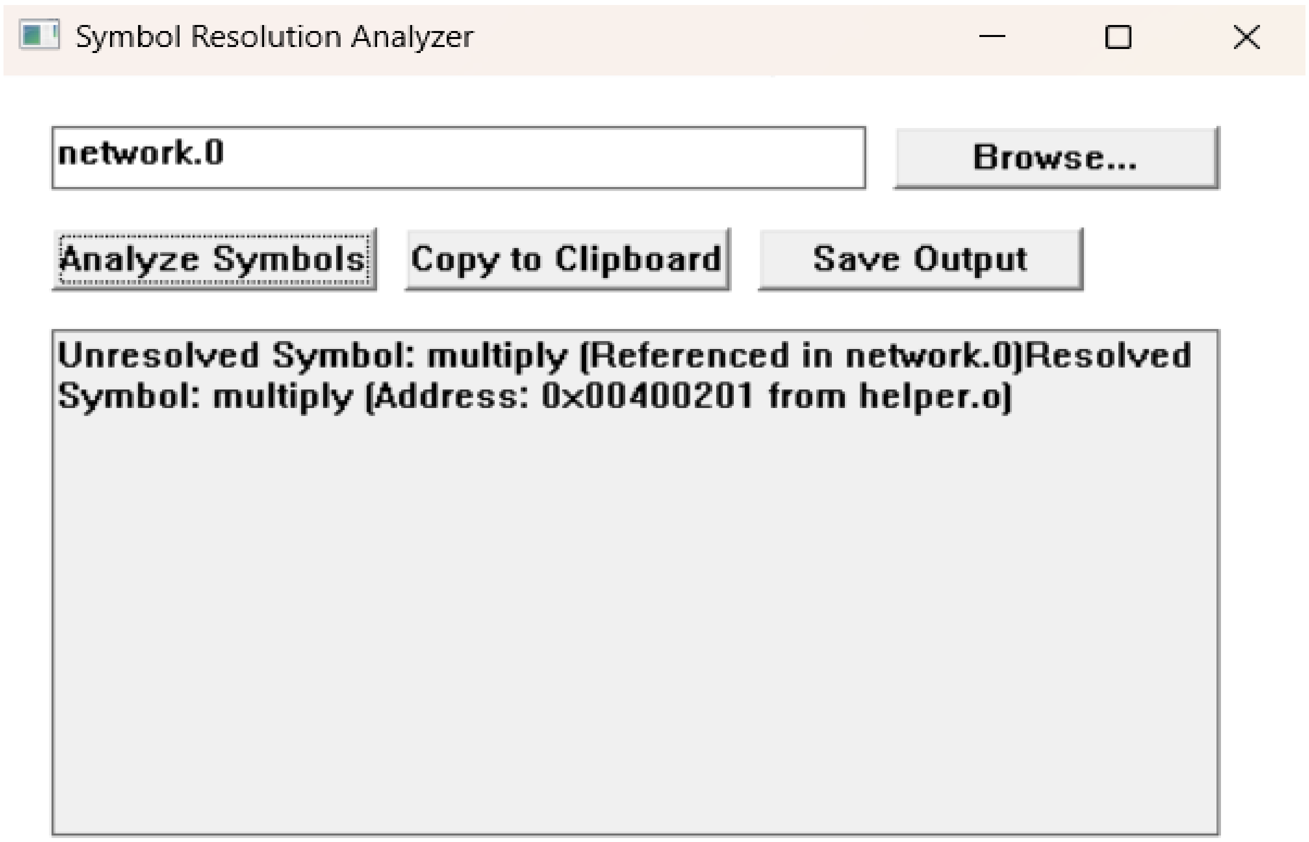
OUTPUT:





```
main.0
```

```
Unresolved Symbol: divide (Referenced in main.0)Resolved
Symbol: divide (Address: 0x00400883 from helper.o)
```



```
Unresolved Symbol: multiply (Referenced in network.0)Resolved
Symbol: multiply (Address: 0x00400201 from helper.o)
```