



EC- 221 OPERATING SYSTEMS

Semester Project Report

Course Instructor: Dr Mehwish Naseer

SUBMITTED BY:

Student Name: Tazeen ur Rehman **Reg #** 466342

Student Name: Rida Zahra **Reg #** 465791

Student Name: Mawa Chaudhary **Reg #** 474121

Student Name: Muhammad Asjad **Reg #** 482857

Degree/ Syndicate: **CE 45 A**

Date: 6th May, 2025

Smart Swapper

Contents

| | |
|----------------------------------|-----------|
| Abstract | 3 |
| Introduction..... | 4 |
| Problem Statement | 5 |
| Objectives | 5 |
| System Overview..... | 10 |
| Software Tools | 10 |
| Working Principle..... | 10 |
| Testing and Results | 13 |
| Challenges Faced | 17 |
| Future Work..... | 18 |
| Conclusion | 18 |
| References..... | 19 |
| Appendix..... | 19 |
| Complete code | 19 |

Abstract

This project report presents the design, implementation, and comparative analysis of three fundamental page replacement algorithms used in operating systems: First-In-First-Out (FIFO), Least Recently Used (LRU), and Optimal page replacement. The simulation is developed in C++ and provides a user-friendly command-line interface that accepts custom reference strings and frame sizes. The primary aim of this project is to evaluate the performance of these algorithms in terms of minimizing page faults—an essential metric for efficient memory management in modern computing environments.

Memory management lies at the core of operating system functionality, especially when systems deal with a larger demand for memory than the available physical RAM. In such cases, virtual memory is used, and page replacement algorithms determine which pages to swap in and out of memory. Poor algorithm selection can lead to frequent page faults, increased disk I/O, and overall degraded performance. Therefore, understanding the working and efficiency of various page replacement techniques is crucial.

The FIFO algorithm replaces the oldest page in memory without considering usage patterns. Although easy to implement, it can perform poorly under certain reference string patterns and is susceptible to Belady's Anomaly. The LRU algorithm, on the other hand, replaces the least recently used page, often yielding better results by more closely approximating real-world usage, but it requires maintaining timestamps or usage history, increasing implementation complexity. The Optimal algorithm, which looks ahead in the reference string to replace the page that won't be used for the longest time, guarantees the least number of page faults but is theoretical, as future references are not known in real-time systems.

In this project, each of these algorithms is implemented and simulated over identical input conditions, with the memory frame contents being printed step-by-step for user clarity. After processing, the number of page faults is reported, and in comparative mode, the best-performing algorithm is identified. The project not only aids in visualizing the dynamic behavior of these memory management policies but also allows students to grasp the trade-offs involved between algorithmic simplicity, real-time feasibility, and performance.

The findings of the project confirm that while the Optimal algorithm performs best in theory, LRU often proves to be a practical and efficient compromise. The FIFO algorithm, despite its ease of implementation, may result in a higher number of page faults. This simulation tool thus serves both as a learning platform and an analytical framework for evaluating memory replacement strategies, providing a foundation for deeper exploration of operating system internals.

Introduction

Efficient memory management is one of the most critical responsibilities of modern operating systems. As applications grow in complexity and data requirements expand, the demand for memory resources often exceeds the capacity of available physical memory (RAM). To address this, operating systems implement virtual memory—a technique that allows processes to use more memory than physically available by swapping pages between RAM and secondary storage (such as a hard disk). However, when a process requests access to a page that is not currently in physical memory, a page fault occurs. The operating system must then decide which page to evict from memory to make room for the new one. This decision is made using page replacement algorithms.

Page replacement algorithms determine which memory pages should be removed when a new page needs to be loaded, and the memory is full. The efficiency of this decision-making process directly impacts system performance, influencing the number of page faults and the speed at which applications run. In real-time systems, embedded applications, and multitasking environments, minimizing page faults is essential to maintaining responsiveness and throughput.

This project focuses on the simulation of three widely studied page replacement algorithms: First-In-First-Out (FIFO), Least Recently Used (LRU), and Optimal. These algorithms are foundational concepts in operating systems courses and represent three different strategies for page replacement. FIFO replaces the oldest page in memory, LRU replaces the page that has not been used for the longest time, and Optimal replaces the page that will not be used for the longest time in the future. Each has its strengths and weaknesses, and their performance varies depending on the reference string and number of available frames.

The goal of this project is not only to implement these algorithms in C++ but also to create an interactive and educational simulation that allows users to input custom reference strings and frame sizes. The simulation displays frame contents at each step, counts page faults, and, when requested, compares the performance of all three algorithms under identical conditions. By visually presenting how memory frames evolve with each page request, the project bridges theoretical understanding with practical application.

Through this project, students gain hands-on experience with core memory management concepts and develop a deeper appreciation for the trade-offs involved in algorithm selection. It also lays the groundwork for exploring more advanced topics such as approximated LRU (e.g., Clock algorithm), demand paging, and working set models. Ultimately, this project highlights the importance of well-designed memory management policies in ensuring the stability, efficiency, and performance of modern computing systems.

Problem Statement

- In systems with limited memory, a process may request more pages than can be held in RAM.
- To manage this, operating systems replace existing pages using page replacement algorithms.
- Choosing an inefficient algorithm may result in excessive page faults, degrading system performance.
- There is a need to analyze and compare popular algorithms under the same conditions to determine which performs best in different scenarios.

Objectives

- Implement Key Algorithms: Develop working C++ implementations of FIFO, LRU, and Optimal page replacement algorithms.
- Enable Custom User Input: Design a user-friendly interface to input any reference string and specify the number of available memory frames.
- Visualize Frame Changes: Display the state of memory frames after each page reference to provide a step-by-step understanding of algorithm behavior.
- Track and Report Page Faults: Accurately count and output the total number of page faults generated by each algorithm.
- Compare Algorithm Performance: Run all three algorithms on the same input and determine which one results in the least number of page faults.
- Reinforce Theoretical Concepts: Strengthen students' understanding of operating system memory management through hands-on simulation and observation.
- Highlight Practical Trade-offs: Show the trade-offs between simplicity (FIFO), implementation complexity (LRU), and theoretical optimality (Optimal).
- Build a Reusable Simulator: Create a modular and reusable simulation tool that can be extended with additional algorithms or features in future coursework or research.
- Support Educational Use: Provide a resource for students and instructors to demonstrate page replacement strategies in classroom settings.

Literature Review

Page replacement algorithms are a cornerstone of operating system design, particularly in systems that utilize virtual memory. These algorithms are responsible for managing the limited physical memory available by deciding which memory pages should be removed to make space for new ones when page faults occur. The choice of algorithm greatly influences system performance and responsiveness, especially in multitasking or memory-constrained environments.

1. The Role of Page Replacement Algorithms

As programs execute, they frequently access various memory pages. In systems with limited RAM, it's impossible to keep all required pages in memory at once. When a page not currently in memory is requested, a page fault occurs, prompting the operating system to bring that page in and possibly evict another. The efficiency with which the operating system handles this replacement process determines the rate of page faults and, consequently, the system's overall performance.

2. First-In-First-Out (FIFO)

FIFO is one of the simplest page replacement strategies. It maintains a queue of pages in the order they were loaded into memory. When a new page must be loaded and memory is full, the oldest page (the one at the front of the queue) is removed, and the new page is added to the rear.

While FIFO is easy to implement, it can perform poorly in certain scenarios. A well-known drawback is Belady's Anomaly, which demonstrates that increasing the number of memory frames can sometimes lead to more page faults, contrary to expectations. This non-intuitive behavior highlights the limitations of FIFO in practical use.

3. Least Recently Used (LRU)

LRU improves upon FIFO by incorporating temporal locality—pages that have been used recently are likely to be used again soon. It replaces the page that has not been accessed for the longest time, assuming that pages used recently will continue to be used in the near future.

LRU generally performs better than FIFO but comes with increased implementation complexity. In a practical system, tracking the exact order of usage can be costly. Approximate LRU algorithms, such as the Clock algorithm, are often used in real operating systems to strike a balance between performance and feasibility.

4. Optimal Page Replacement

The Optimal algorithm provides the best possible performance by replacing the page that will not be used for the longest time in the future. It requires knowledge of future memory accesses, making it impractical for real-time implementation. However, it is frequently used in academic and research settings as a benchmark to evaluate the efficiency of other algorithms.

By comparing FIFO and LRU against Optimal, we can assess how closely real-world algorithms approximate theoretical best-case performance. This comparative analysis helps guide system designers in selecting or designing appropriate memory management strategies.

5. Relevance in Modern Operating Systems

Modern operating systems, such as Linux, Windows, and macOS, often use variations of LRU for managing virtual memory. For example, Linux implements a modified version of LRU using two lists: one for active pages and one for inactive pages, balancing performance and system responsiveness. These algorithms are fine-tuned to handle diverse workloads, background processes, and multi-user environments.

6. Educational Importance

For students and practitioners, understanding these algorithms is fundamental to grasping how memory management works under the hood. Simulating page replacement strategies enables learners to visualize how different algorithms behave under various inputs and constraints. This hands-on approach is essential in bridging the gap between theoretical understanding and real-world application.

Proposed Solution and Contribution

Various solutions have been implemented in real-world operating systems and educational tools to address the problem of efficient memory management through page replacement. These existing systems demonstrate a range of strategies—from simple queue-based algorithms to more advanced, approximation-based techniques optimized for performance.

1. Page Replacement in Real Operating Systems

Modern operating systems like Linux, Windows, and macOS use highly optimized page replacement strategies based on empirical observations and approximations of theoretical algorithms:

Linux uses a variant of the Least Recently Used (LRU) algorithm, specifically a two-list system (active and inactive lists). It tries to approximate LRU without the heavy overhead of tracking every page access.

Windows OS employs a combination of working set-based page management and priority classes, where each process has a certain number of pages it is allowed to hold in memory. When the working set is full, the system selects a page to replace, typically using a strategy similar to LRU.

macOS uses a dynamic pager that swaps inactive pages to disk and brings in active pages as needed, leveraging usage history and memory pressure indicators.

These systems do not use pure FIFO or Optimal algorithms due to performance, feasibility, and overhead concerns but may incorporate FIFO-like behavior for background processes or seldom-used data.

2. Simulation Tools and Teaching Aid

Several educational tools have been developed to simulate page replacement algorithms and help students visualize memory behavior:

OS simulators like EduOS, Little OS, or Nachos allow learners to experiment with different memory management techniques.

Web-based simulators such as the ones on GeeksforGeeks and TutorialsPoint offer visual demonstrations of FIFO, LRU, and Optimal algorithms using predefined or user-provided inputs.

University assignments and online judges often include page replacement simulation tasks to reinforce memory management concepts in a controlled environment.

While these tools are effective in teaching, they often abstract away the implementation details, depriving students of hands-on coding experience. Furthermore, many simulators only demonstrate algorithm behavior visually and do not support in-depth comparison, performance measurement, or detailed logging.

3. Limitations of Existing Solutions

Despite their usefulness, current solutions often have the following limitations:

Lack of Customization: Many online simulators do not allow arbitrary-length reference strings or detailed control over simulation steps.

No Comparative Analysis: They simulate one algorithm at a time, without side-by-side comparisons of multiple strategies under the same input.

Limited Feedback: Most tools provide page fault counts but lack real-time frame visualization or step-wise tracing.

These gaps highlight the need for a customizable, console-based simulator that not only implements FIFO, LRU, and Optimal algorithms from scratch but also offers detailed feedback and performance evaluation—all of which this project aims to provide.

Graphical User Interface (GUI)

The project features a user-friendly Graphical User Interface (GUI) developed using Windows Forms in C++/CLI. The interface is designed to provide an intuitive and interactive way for users to input parameters, select algorithms, and visualize results with minimal complexity. The GUI significantly enhances user engagement compared to traditional command-line tools, making the simulation more accessible, especially for non-programmers and beginners in operating systems.

Key Components of the GUI:

Input Display Panel: Shows the number of frames, total pages, and the reference string in a clean, formatted manner for user clarity.

Algorithm Selection Dropdown: Allows users to choose from FIFO, LRU, Optimal, or All Algorithms through a ComboBox. This makes testing and comparison seamless and avoids re-entering data.

RUN Button: Triggers the execution of the selected algorithm and updates the results dynamically on the screen. It provides immediate feedback and reduces the complexity of running simulations.

Result Labels: Display total page faults and the best-performing algorithm when all are run. The results are highlighted using color and font styling for better visibility.

Show Details Button: When clicked, it reveals a detailed, tabbed DataGridView section for step-by-step memory frame visualization, including each step, frame contents, and page fault indication.

Tabbed Data Visualization: For each selected algorithm, a tab page is created showing frame-by-frame changes in a table format, including clear indicators of page faults and highlighted entries using colors and font weights.

Dynamic Layout: The form dynamically resizes based on user interaction, such as revealing or hiding detailed views, which enhances flexibility and maintains a clean interface.

Visual Enhancements:

A consistent pastel color scheme improves readability and aesthetics.

Bold fonts and clear layout separation make data easy to interpret.

Alternating row colors in the DataGridView improve visual tracking of page states.

System Overview

The simulation has three core components:

- **User Interface:** Accepts reference string and frame count.
- **Algorithm Implementations:** FIFO, LRU (with timestamp tracking), and Optimal (using future prediction).

Software Tools

- **Programming Language:** C++
- **Compiler:** g++ or any standard C++ compiler
- **IDE/Platform:** Visual Studio / Code::Blocks / Terminal
- **Libraries Used:** <vector>, <iostream>, <limits>, <algorithm>

Working Principle

The core functionality of the simulator revolves around the execution of three distinct page replacement algorithms—FIFO, LRU, and Optimal. Each algorithm processes a given page reference string and simulates how memory frames are managed as pages are loaded, accessed, or replaced. The simulator is designed to visually demonstrate the changes in frame content after each page access, while also tracking the number of page faults encountered during execution.

1. User Input and Initialization

- The simulation begins by prompting the user to enter:
- The total number of pages in the reference string.
- The page reference string itself (as a sequence of integers).
- The number of available memory frames.
- The choice of which algorithm(s) to execute.

The system validates input to ensure correctness and robustness, preventing invalid entries such as negative numbers or non-numeric characters.

2. FIFO Algorithm Workflow

A vector is used to represent memory frames, initialized to a default invalid value (e.g., -1).

The algorithm uses a simple index pointer (like a circular queue) to identify which frame to replace.

When a page fault occurs (i.e., the requested page is not found in memory), the oldest page—pointed to by the index—is replaced with the new page.

The index is then incremented (modulo total frames) to cycle through frame positions.

Characteristics:

- Simple implementation.
- Replaces the earliest-loaded page.
- Susceptible to Belady's Anomaly

3. LRU Algorithm Workflow

- Alongside the frame vector, a lastUsed vector maintains timestamps representing the last access time of each frame.
- For each page request:
 - If the page is already present, its timestamp is updated.
 - If it is not present:
 - A page fault is recorded.
 - If a free frame exists, the page is placed there.
 - Otherwise, the page with the oldest timestamp (least recently used) is evicted and replaced.
- The global simulation time is incremented with every page access to simulate recency tracking

Characteristics:

- Better performance in practice.
- Mimics user behavior with temporal locality.
- Requires additional overhead for timestamp tracking.

4. Optimal Algorithm Workflow

- Maintains a vector of current frames.
- For each page:
- If it exists in memory, it is considered a hit.
- If not, a page fault occurs.
- If space is available, the page is simply added.
- If no space is available, the algorithm predicts which currently loaded page will not be used for the longest time in the future by scanning the remaining reference string.
- That page is replaced with the current one.
- The predict() function is used to determine the best candidate for replacement based on future usage

Characteristics:

- Theoretically optimal (least number of faults).
- Not practical in real systems due to its reliance on future knowledge.
- Used for benchmarking.

5. Output Generation

- After each page reference, the program:
- Prints the current page.
- Displays the contents of the memory frames.
- Indicates whether a page fault occurred.
- At the end of the simulation, the total number of page faults is displayed for the selected algorithm.
- If the user chooses to run all three algorithms in a single run:
- A summary is shown with total page faults from each algorithm.
- The simulator automatically identifies and displays the most efficient algorithm for the given input

6. Looping and Exit

- After execution, the program prompts the user whether they want to run another simulation.
- If the user chooses 'y', the loop restarts; if 'n', the program exits gracefully.

Testing and Results

The code was tested using multiple reference strings:

Example: 7 0 1 2 0 3 0 4 2 3 0 3 2 with 3 frames
Results:

FIFO Page Faults: 10

LRU Page Faults: 8

Optimal Page Faults: 7

This confirmed that Optimal always performs best, followed by LRU, and FIFO.

Output:



Smart Swapper - Page Replacement

SMART SWAPPER

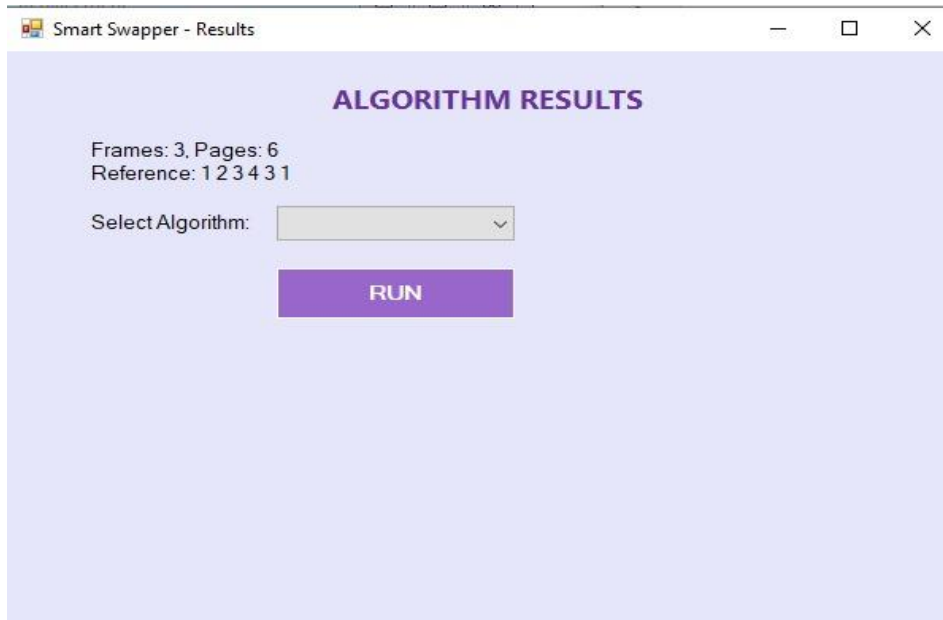
Frames:

Pages:

Reference String:

Reference string must contain exactly 6 numbers.

SUBMIT



Smart Swapper - Results

ALGORITHM RESULTS

Frames: 3, Pages: 6
Reference: 1 2 3 4 3 1

Select Algorithm:

RUN

Smart Swapper - Results

ALGORITHM RESULTS

Frames: 3, Pages: 6
Reference: 1 2 3 4 3 1

Select Algorithm:

FIFO Page Faults: 5

FIFO

| Step | Reference | Frame 1 | Frame 2 | Frame 3 | Page Fault |
|------|-----------|---------|---------|---------|------------|
| 0 | 1 | 1 * | - | - | Yes |
| 1 | 2 | 1 | 2 * | - | Yes |
| 2 | 3 | 1 | 2 | 3 * | Yes |
| 3 | 4 | 4 * | 2 | 3 | Yes |
| 4 | 3 | 4 | 2 | 3 | No |
| 5 | 1 | 4 | 1 * | 3 | Yes |

Smart Swapper - Results

ALGORITHM RESULTS

Frames: 3, Pages: 6
Reference: 1 2 3 4 3 1

Select Algorithm:

LRU Page Faults: 5

LRU

| Step | Reference | Frame 1 (Last Used) | Frame 2 (Last Used) | Frame 3 (Last Used) |
|------|-----------|---------------------|---------------------|---------------------|
| 0 | 1 | 1 (1) * | - | - |
| 1 | 2 | 1 (1) | 2 (2) * | - |
| 2 | 3 | 1 (1) | 2 (2) | 3 (3) * |
| 3 | 4 | 4 (4) * | 2 (2) | 3 (3) |
| 4 | 3 | 4 (4) | 2 (2) | 3 (5) |
| 5 | 1 | 4 (4) | 1 (6) * | 3 (5) |

Smart Swapper - Results

ALGORITHM RESULTS

Frames: 3, Pages: 6
Reference: 1 2 3 4 3 1

Select Algorithm: Optimal

RUN HIDE DETAILS

Optimal Page Faults: 4

Optimal

| Frame 1 (Next Use) | Frame 2 (Next Use) | Frame 3 (Next Use) | Page Fault |
|--------------------|--------------------|--------------------|------------|
| 1 (5) * | - | - | Yes |
| 1 (5) | 2 (never) * | - | Yes |
| 1 (5) | 2 (never) | 3 (4) * | Yes |
| 1 (5) | 4 (never) * | 3 (4) | Yes |
| 1 (5) | 4 (never) | 3 (never) | No |
| 1 (never) | 4 (never) | 3 (never) | No |

Smart Swapper - Results

ALGORITHM RESULTS

Frames: 3, Pages: 6
Reference: 1 2 3 4 3 1

Select Algorithm: All Algorithms

RUN HIDE DETAILS

FIFO: 5, LRU: 5, Optimal: 4

Best Algorithm: Optimal

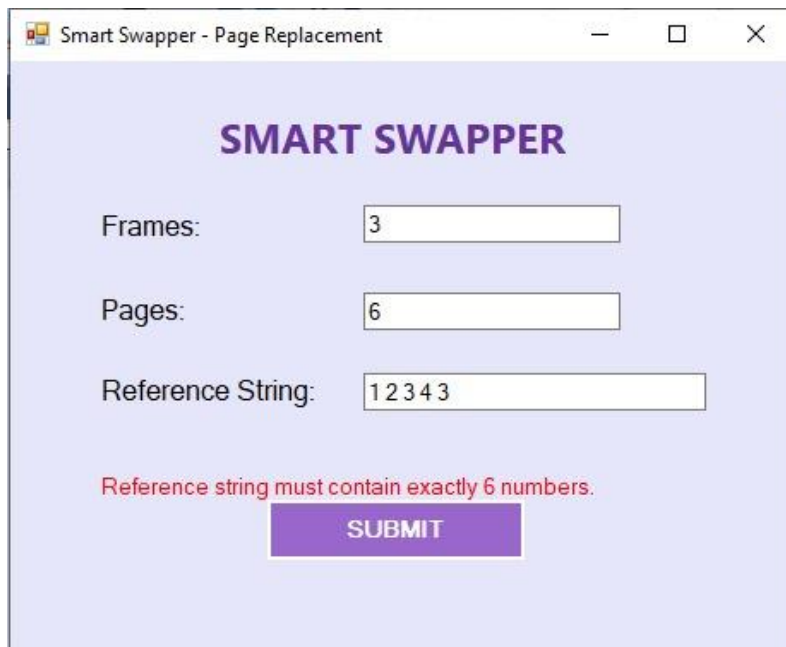
FIFO LRU Optimal

| Step | Reference | Frame 1 | Frame 2 | Frame 3 | Page Fault |
|------|-----------|---------|---------|---------|------------|
| 0 | 1 | 1 * | - | - | Yes |
| 1 | 2 | 1 | 2 * | - | Yes |
| 2 | 3 | 1 | 2 | 3 * | Yes |
| 3 | 4 | 4 * | 2 | 3 | Yes |
| 4 | 3 | 4 | 2 | 3 | No |
| 5 | 1 | 4 | 1 * | 3 | Yes |

Error Handling:



The screenshot shows a web browser window titled "Smart Swapper - Page Replacement". The page has a light blue background and the title "SMART SWAPPER" in bold purple text. There are three input fields: "Frames:" with the value "a", "Pages:" which is empty, and "Reference String:" which is empty. Below the input fields, a red error message reads "Invalid input! Input string was not in a correct format." A purple "SUBMIT" button is located at the bottom.



The screenshot shows the same "Smart Swapper - Page Replacement" web browser window. The "Frames:" field now contains the value "3", the "Pages:" field contains "6", and the "Reference String:" field contains "1 2 3 4 3". A red error message below the fields reads "Reference string must contain exactly 6 numbers." The purple "SUBMIT" button remains at the bottom.

Challenges Faced

- **Handling invalid input:** Added input validation for negative values and non-integers.
- **LRU implementation:** Required managing timestamps manually using arrays.
- **Optimal algorithm:** Required a prediction function to scan future references.
- **Console formatting:** Ensured that outputs were cleanly formatted for clarity.

Future Work

- Add support for real-time graphical visualization using SDL or SFML.
- Include other algorithms like Clock, LFU, or NRU for broader comparison.
- Export result logs to a file for further analysis.
- Create a web-based version using JavaScript for accessibility.

Conclusion

This project successfully demonstrates the implementation and comparative analysis of three essential page replacement algorithms—FIFO, LRU, and Optimal—using the C++ programming language. Through a step-by-step simulation of memory management operations, users gain hands-on experience with the inner workings of virtual memory and the importance of intelligent page replacement strategies.

The simulation effectively highlights the operational principles, advantages, and limitations of each algorithm. FIFO, though simple and easy to implement, can result in suboptimal performance due to its inability to consider recent page usage. LRU provides better results by leveraging temporal locality but requires additional data structures to track usage history. The Optimal algorithm, while theoretical in nature due to its reliance on future knowledge, consistently delivers the lowest page fault count and serves as a benchmark for evaluating other strategies.

The project not only facilitates theoretical learning but also enhances practical understanding through real-time visualization and performance tracking. By allowing users to input custom reference strings and frame sizes, the simulator accommodates a wide range of test cases and usage patterns, offering insights into how algorithm behavior changes under different workloads.

Moreover, the inclusion of a comparison module that identifies the most efficient algorithm based on the simulation results adds an analytical dimension to the project. It encourages users to think critically about memory management design decisions in real-world operating systems.

In conclusion, this project bridges the gap between classroom theory and system-level programming. It serves as an educational tool for understanding core OS concepts and lays a solid foundation for future enhancements such as GUI development, implementation of additional algorithms like Clock or LFU, and real-time integration with system memory data. Ultimately, the project reinforces the critical role that page replacement algorithms play in ensuring efficient, stable, and responsive computing environments.

References

1. Silberschatz, A., Galvin, P. B., & Gagne, G. (2020). *Operating System Concepts*. Wiley.
2. Tanenbaum, A. S., & Bos, H. (2015). *Modern Operating Systems*. Pearson.
3. GeeksforGeeks: Page Replacement Algorithms
4. TutorialsPoint: Virtual Memory and Page Replacement
5. Operating Systems course lectures (NUST CEME, Spring 2025)

Appendix

Complete code

Form21.h

```
#pragma once
#include <iostream>
#include <vector>
#include <limits>
#include <algorithm>
#include <array>
#include <iterator> // for distance

using namespace std;
using namespace System::Drawing;
using namespace System::Windows::Forms;

namespace Project1 {
    using namespace System;
    using namespace System::Windows::Forms;
    using namespace System::Collections::Generic;

    public ref class Form2 : public Form {
    private:
        int frames;
        int pages;
        List<int>^ refString;
        DataGridView^ dataGridViewDetails;
        TabControl^ tabControlDetails;

    public:
```

```

Form2(int f, int p, List<int>^ refStr) {
    InitializeComponent();
    frames = f;
    pages = p;
    refString = refStr;

    // Updated label with better formatting
    labelInput->Text = String::Format("Frames: {0}, Pages: {1}\nReference: {2}",
        frames, pages, String::Join(" ", refStr));

    InitializeAdditionalComponents();
}

protected:
    ~Form2() {
        if (components) delete components;
    }

private:
    // Controls
    ComboBox^ comboBoxAlgorithm;
    Button^ buttonRun;
    Button^ buttonDetails;
    Label^ labelAlgorithm;
    Label^ labelOutput;
    Label^ labelResult;
    Label^ labelInput;
    Label^ labelHeader;
    System::ComponentModel::Container^ components;

    void InitializeComponent(void) {
        // Main form setup
        this->SuspendLayout();
        this->BackColor = System::Drawing::Color::FromArgb(230, 230, 250);
        this->ClientSize = System::Drawing::Size(600, 400);
        this->Text = L"Smart Swapper - Results";

        // Header label
        this->labelHeader = gcnew Label();
        this->labelHeader->AutoSize = true;
        this->labelHeader->Font = gcnew System::Drawing::Font(L"Segoe UI", 14,
FontStyle::Bold);
        this->labelHeader->ForeColor = System::Drawing::Color::FromArgb(102, 51, 153);

```

```

this->labelHeader->Location = System::Drawing::Point(200, 20);
this->labelHeader->Text = L"ALGORITHM RESULTS";
this->Controls->Add(labelHeader);

// Input label
this->labelInput = gnew Label();
this->labelInput->Font = gnew System::Drawing::Font(L"Microsoft Sans Serif", 9.75F);
this->labelInput->Location = System::Drawing::Point(50, 60);
this->labelInput->Size = System::Drawing::Size(500, 40);
this->Controls->Add(labelInput);

// Algorithm selection
this->labelAlgorithm = gnew Label();
this->labelAlgorithm->Font = gnew System::Drawing::Font(L"Microsoft Sans Serif",
9.75F);
this->labelAlgorithm->Location = System::Drawing::Point(50, 110);
this->labelAlgorithm->Text = L"Select Algorithm:";
this->labelAlgorithm->AutoSize = true;
this->Controls->Add(labelAlgorithm);

this->comboBoxAlgorithm = gnew ComboBox();
this->comboBoxAlgorithm->DropDownStyle = ComboBoxStyle::DropDownList;
this->comboBoxAlgorithm->Font = gnew System::Drawing::Font(L"Microsoft Sans
Serif", 9.75F);
this->comboBoxAlgorithm->Items->AddRange(gnew cli::array<Object^>(4) {
    "FIFO", "LRU", "Optimal", "All Algorithms"
});
this->comboBoxAlgorithm->Location = System::Drawing::Point(170, 107);
this->comboBoxAlgorithm->Size = System::Drawing::Size(150, 24);
this->comboBoxAlgorithm->BackColor = System::Drawing::Color::White;
this->Controls->Add(comboBoxAlgorithm);

// Run button
this->buttonRun = gnew Button();
this->buttonRun->BackColor = System::Drawing::Color::FromArgb(153, 102, 204);
this->buttonRun->FlatStyle = FlatStyle::Flat;
this->buttonRun->Font = gnew System::Drawing::Font(L"Microsoft Sans Serif", 9.75F,
FontStyle::Bold);
this->buttonRun->ForeColor = System::Drawing::Color::White;
this->buttonRun->Location = System::Drawing::Point(170, 150);
this->buttonRun->Size = System::Drawing::Size(150, 35);
this->buttonRun->Text = L"RUN";
this->buttonRun->Click += gnew EventHandler(this, &Form2::buttonRun_Click);

```

```

        this->Controls->Add(buttonRun);

        // Results labels
        this->labelOutput = gcnew Label();
        this->labelOutput->Font = gcnew System::Drawing::Font(L"Microsoft Sans Serif", 9.75F);
        this->labelOutput->Location = System::Drawing::Point(50, 200);
        this->labelOutput->AutoSize = true;
        this->Controls->Add(labelOutput);

        this->labelResult = gcnew Label();
        this->labelResult->Font = gcnew System::Drawing::Font(L"Microsoft Sans Serif", 9.75F,
FontStyle::Bold);
        this->labelResult->ForeColor = System::Drawing::Color::FromArgb(0, 128, 0);
        this->labelResult->Location = System::Drawing::Point(50, 230);
        this->labelResult->AutoSize = true;
        this->Controls->Add(labelResult);

        this->ResumeLayout(false);
    }

    void InitializeAdditionalComponents() {
        // Details button
        this->buttonDetails = gcnew Button();
        this->buttonDetails->BackColor = System::Drawing::Color::FromArgb(153, 102, 204);
        this->buttonDetails->FlatStyle = FlatStyle::Flat;
        this->buttonDetails->Font = gcnew System::Drawing::Font(L"Microsoft Sans Serif",
9.75F, FontStyle::Bold);
        this->buttonDetails->ForeColor = System::Drawing::Color::White;
        this->buttonDetails->Location = System::Drawing::Point(350, 150);
        this->buttonDetails->Size = System::Drawing::Size(150, 35);
        this->buttonDetails->Text = L"SHOW DETAILS";
        this->buttonDetails->Click += gcnew EventHandler(this, &Form2::buttonDetails_Click);
        this->buttonDetails->Visible = false; // Initially hidden
        this->Controls->Add(buttonDetails);

        // Tab control for algorithm details
        this->tabControlDetails = gcnew TabControl();
        this->tabControlDetails->Location = Point(50, 270);
        this->tabControlDetails->Size = System::Drawing::Size(500, 300);
        this->tabControlDetails->Visible = false;
        this->Controls->Add(tabControlDetails);
    }

```

```

void buttonRun_Click(Object^ sender, EventArgs^ e) {
    if (comboBoxAlgorithm->SelectedItem == nullptr) {
        labelOutput->ForeColor = Color::Red;
        labelOutput->Text = "Please select an algorithm!";
        return;
    }

    // Convert managed list to native array
    int* ref_arr = new int[pages];
    for (int i = 0; i < pages; ++i)
        ref_arr[i] = refString[i];

    String^ selected = comboBoxAlgorithm->SelectedItem->ToString();
    int fifoFaults = 0, lruFaults = 0, optFaults = 0;

    if (selected == "FIFO") {
        fifoFaults = simulateFIFO(ref_arr, pages, frames);
        labelOutput->ForeColor = Color::Black;
        labelOutput->Text = "FIFO Page Faults: " + fifoFaults.ToString();
        labelResult->Text = "";
    }
    else if (selected == "LRU") {
        lruFaults = simulateLRU(ref_arr, pages, frames);
        labelOutput->ForeColor = Color::Black;
        labelOutput->Text = "LRU Page Faults: " + lruFaults.ToString();
        labelResult->Text = "";
    }
    else if (selected == "Optimal") {
        optFaults = simulateOptimal(ref_arr, pages, frames);
        labelOutput->ForeColor = Color::Black;
        labelOutput->Text = "Optimal Page Faults: " + optFaults.ToString();
        labelResult->Text = "";
    }
    else if (selected == "All Algorithms") {
        fifoFaults = simulateFIFO(ref_arr, pages, frames);
        lruFaults = simulateLRU(ref_arr, pages, frames);
        optFaults = simulateOptimal(ref_arr, pages, frames);

        labelOutput->ForeColor = Color::Black;
        labelOutput->Text = String::Format("FIFO: {0}, LRU: {1}, Optimal: {2}",
            fifoFaults, lruFaults, optFaults);

        String^ best;
    }
}

```

```

        int minFaults = min({ fifoFaults, lruFaults, optFaults });
        if (minFaults == optFaults) best = "Optimal";
        else if (minFaults == lruFaults) best = "LRU";
        else best = "FIFO";

        labelResult->Text = "Best Algorithm: " + best;
    }

    delete[] ref_arr;

    // Show details button after running
    buttonDetails->Visible = true;
    this->ClientSize = System::Drawing::Size(600, 400);
}

void buttonDetails_Click(Object^ sender, EventArgs^ e) {
    if (tabControlDetails->Visible) {
        tabControlDetails->Visible = false;
        this->ClientSize = System::Drawing::Size(600, 400);
        buttonDetails->Text = "SHOW DETAILS";
        tabControlDetails->TabPage->Clear();
    }
    else {
        // Convert managed list to native array
        int* ref_arr = new int[pages];
        for (int i = 0; i < pages; ++i)
            ref_arr[i] = refString[i];

        String^ selected = comboBoxAlgorithm->SelectedItem->ToString();
        tabControlDetails->TabPage->Clear();

        if (selected == "FIFO") {
            CreateDetailsTab("FIFO", GenerateFIFODetails(ref_arr, pages, frames));
        }
        else if (selected == "LRU") {
            CreateDetailsTab("LRU", GenerateLRUDetails(ref_arr, pages, frames));
        }
        else if (selected == "Optimal") {
            CreateDetailsTab("Optimal", GenerateOptimalDetails(ref_arr, pages, frames));
        }
        else if (selected == "All Algorithms") {
            CreateDetailsTab("FIFO", GenerateFIFODetails(ref_arr, pages, frames));
            CreateDetailsTab("LRU", GenerateLRUDetails(ref_arr, pages, frames));
        }
    }
}

```



```

        CreateDetailsTab("Optimal", GenerateOptimalDetails(ref_arr, pages, frames));
    }

    delete[] ref_arr;

    tabControlDetails->Visible = true;
    this->ClientSize = System::Drawing::Size(600, 600);
    buttonDetails->Text = "HIDE DETAILS";
}
}

void CreateDetailsTab(String^ algorithmName, List<List<String^>^>^ details) {
   TabPage^ tabPage = gcnew TabPage(algorithmName);
    DataGridView^ dataGridView = gcnew DataGridView();
    dataGridView->Dock = DockStyle::Fill;
    dataGridView->ReadOnly = true;
    dataGridView->AllowUserToAddRows = false;
    dataGridView->AllowUserToDeleteRows = false;
    dataGridView->AllowUserToResizeRows = false;
    dataGridView->AutoSizeColumnsMode
DataGridViewAutoSizeColumnsMode::AllCells;
    dataGridView->RowHeadersVisible = false;
    dataGridView->BackgroundColor = Color::White;

    // Add columns first
    for (int i = 0; i < details[0]->Count; i++)
    {
        DataGridViewTextBoxColumn^ column = gcnew DataGridViewTextBoxColumn();
        column->HeaderText = details[0][i]; // Set header text immediately
        column->Name = "Column" + i;      // Give each column a unique name
        column->SortMode = DataGridViewColumnSortMode::NotSortable;
        dataGridView->Columns->Add(column);
    }

    // Add rows (skip header row)
    for (int i = 1; i < details->Count; i++)
    {
        // Create a new row
        DataGridViewRow^ row = gcnew DataGridViewRow();
        row->CreateCells(dataGridView);

        // Fill the row with data
        for (int j = 0; j < details[i]->Count; j++)

```

```

        {
            row->Cells[j]->Value = details[i][j];

            // Highlight page faults if this is the last column
            if (j == details[i]->Count - 1 && details[i][j] == "Yes")
            {
                row->Cells[j]->Style->ForeColor = Color::Red;
                row->Cells[j]->Style->Font = gcnew System::Drawing::Font(
                    dataGridView->Font, FontStyle::Bold);
            }
        }

        // Add the completed row to the grid
        dataGridView->Rows->Add(row);
    }

    // Auto-resize columns to fit content
    dataGridView->AutoSizeColumns(DataGridViewAutoSizeColumnsMode::AllCells);
    // Style the grid
    dataGridView->DefaultCellStyle->Font = gcnew System::Drawing::Font("Microsoft Sans
Serif", 9);
    dataGridView->ColumnHeadersDefaultCellStyle->Font = gcnew
System::Drawing::Font("Microsoft Sans Serif", 9, FontStyle::Bold);
    dataGridView->AlternatingRowsDefaultCellStyle->BackColor = Color::FromArgb(240,
240, 240);

    tabPage->Controls->Add(dataGridView);
    tabControlDetails->TabPage->Add(tabPage);
}

List<List<String^>>^ GenerateFIFODetails(const int ref_str[], int total_pages, int
total_frames) {
    List<List<String^>>^ details = gcnew List<List<String^>>^>();
    vector<int> frames(total_frames, -1);
    int index = 0;

    // Create header row
    List<String^>^ header = gcnew List<String^>();
    header->Add("Step");
    header->Add("Reference");
    for (int i = 0; i < total_frames; i++) {
        header->Add("Frame " + (i + 1).ToString());
    }
}

```

```

header->Add("Page Fault");
details->Add(header);

for (int i = 0; i < total_pages; ++i) {
    List<String^>^ row = gcnew List<String^>();
    row->Add(i.ToString());
    row->Add(ref_str[i].ToString());

    bool pageFault = find(frames.begin(), frames.end(), ref_str[i]) == frames.end();
    if (pageFault) {
        frames[index % total_frames] = ref_str[i];
        index++;
    }

    // Add frame contents
    for (int j = 0; j < total_frames; ++j) {
        if (frames[j] == -1) {
            row->Add("-");
        }
        else {
            row->Add(frames[j].ToString() + (frames[j] == ref_str[i] && pageFault ? " *" : ""));
        }
    }

    row->Add(pageFault ? "Yes" : "No");
    details->Add(row);
}

return details;
}

List<List<String^>^>^ GenerateLRUDetails(const int ref_str[], int total_pages, int
total_frames) {
    List<List<String^>^>^ details = gcnew List<List<String^>^>();
    vector<int> frames(total_frames, -1);
    vector<int> lastUsed(total_frames, -1);
    int time = 0;

    // Create header row
    List<String^>^ header = gcnew List<String^>();
    header->Add("Step");
    header->Add("Reference");
    for (int i = 0; i < total_frames; i++) {

```

```

        header->Add("Frame " + (i + 1).ToString() + " (Last Used)");
    }
    header->Add("Page Fault");
    details->Add(header);

    for (int i = 0; i < total_pages; ++i) {
        time++;
        List<String>^ row = gcnew List<String>();
        row->Add(i.ToString());
        row->Add(ref_str[i].ToString());

        auto it = find(frames.begin(), frames.end(), ref_str[i]);
        bool pageFault = it == frames.end();

        if (!pageFault) {
            lastUsed[it - frames.begin()] = time;
        }
        else {
            int empty = find(frames.begin(), frames.end(), -1) - frames.begin();
            if (empty < total_frames) {
                frames[empty] = ref_str[i];
                lastUsed[empty] = time;
            }
            else {
                int lru = min_element(lastUsed.begin(), lastUsed.end()) - lastUsed.begin();
                frames[lru] = ref_str[i];
                lastUsed[lru] = time;
            }
        }
    }

    // Add frame contents
    for (int j = 0; j < total_frames; ++j) {
        if (frames[j] == -1) {
            row->Add("-");
        }
        else {
            row->Add(frames[j].ToString() + " (" + lastUsed[j].ToString() + ") " +
                (frames[j] == ref_str[i] && pageFault ? " * " : ""));
        }
    }

    row->Add(pageFault ? "Yes" : "No");
    details->Add(row);

```

```

    }

    return details;
}

List<List<String^>^> GenerateOptimalDetails(const int ref_str[], int total_pages, int
total_frames) {
    List<List<String^>^> details = gcnew List<List<String^>^>();
    vector<int> frames;

    // Create header row
    List<String^> header = gcnew List<String^>();
    header->Add("Step");
    header->Add("Reference");
    for (int i = 0; i < total_frames; i++) {
        header->Add("Frame " + (i + 1).ToString() + " (Next Use)");
    }
    header->Add("Page Fault");
    details->Add(header);

    for (int i = 0; i < total_pages; ++i) {
        List<String^> row = gcnew List<String^>();
        row->Add(i.ToString());
        row->Add(ref_str[i].ToString());

        bool pageFault = find(frames.begin(), frames.end(), ref_str[i]) == frames.end();

        if (pageFault) {
            if (frames.size() < total_frames) {
                frames.push_back(ref_str[i]);
            }
            else {
                int idx = predict(ref_str, total_pages, frames, i + 1);
                frames[idx] = ref_str[i];
            }
        }

        // Add frame contents
        for (int j = 0; j < total_frames; ++j) {
            if (j >= frames.size()) {
                row->Add("-");
                continue;
            }

```

```

        // Find next use
        int nextUse = -1;
        for (int k = i + 1; k < total_pages; ++k) {
            if (ref_str[k] == frames[j]) {
                nextUse = k;
                break;
            }
        }

        String^ nextUseStr = nextUse == -1 ? "never" : nextUse.ToString();
        row->Add(frames[j].ToString() + " (" + nextUseStr + ")" +
            (frames[j] == ref_str[i] && pageFault ? " *" : ""));
    }

    row->Add(pageFault ? "Yes" : "No");
    details->Add(row);
}

return details;
}

int simulateFIFO(const int ref_str[], int total_pages, int total_frames) {
    vector<int> frames(total_frames, -1);
    int page_faults = 0, index = 0;
    for (int i = 0; i < total_pages; ++i) {
        if (find(frames.begin(), frames.end(), ref_str[i]) == frames.end()) {
            frames[index % total_frames] = ref_str[i];
            index++;
            page_faults++;
        }
    }
    return page_faults;
}

int simulateLRU(const int ref_str[], int total_pages, int total_frames) {
    vector<int> frames(total_frames, -1);
    vector<int> lastUsed(total_frames, -1);
    int page_faults = 0;
    int time = 0;

    for (int i = 0; i < total_pages; ++i) {
        time++;

```

```

    auto it = find(frames.begin(), frames.end(), ref_str[i]);

    if (it != frames.end()) {
        // Page found in frames
        lastUsed[distance(frames.begin(), it)] = time;
    }
    else {
        // Page fault
        page_faults++;
        auto empty_it = find(frames.begin(), frames.end(), -1);

        if (empty_it != frames.end()) {
            // Empty frame available
            size_t empty_pos = distance(frames.begin(), empty_it);
            frames[empty_pos] = ref_str[i];
            lastUsed[empty_pos] = time;
        }
        else {
            // Need to replace a page
            size_t lru_pos = distance(lastUsed.begin(),
                min_element(lastUsed.begin(), lastUsed.end()));
            frames[lru_pos] = ref_str[i];
            lastUsed[lru_pos] = time;
        }
    }
}
return page_faults;
}

int predict(const int ref_str[], int total_pages, const vector<int>& frames, int index) {
    int result = -1, farthest = index;
    for (int i = 0; i < frames.size(); i++) {
        int j;
        for (j = index; j < total_pages; j++) {
            if (frames[i] == ref_str[j]) {
                if (j > farthest) {
                    farthest = j;
                    result = i;
                }
                break;
            }
        }
    }
    if (j == total_pages) return i;
}

```

```
    }
    return result == -1 ? 0 : result;
}

int simulateOptimal(const int ref_str[], int total_pages, int total_frames) {
    vector<int> frames;
    int page_faults = 0;
    for (int i = 0; i < total_pages; ++i) {
        if (find(frames.begin(), frames.end(), ref_str[i]) != frames.end())
            continue;
        if (frames.size() < total_frames)
            frames.push_back(ref_str[i]);
        else {
            int idx = predict(ref_str, total_pages, frames, i + 1);
            frames[idx] = ref_str[i];
        }
        page_faults++;
    }
    return page_faults;
}
};
}
```

Form1.h

```
#pragma once
#include "Form21.h"

namespace Project1 {

    public ref class Form1 : public
    System::Windows::Forms::Form
    {
    public:
        Form1(void)
        {
            InitializeComponent();
        }

    protected:
        ~Form1()
    }
```



```

    {
        if (components)
            delete components;
    }

private:
    // Controls declaration
    System::Windows::Forms::Label^ labelFrames;
    System::Windows::Forms::TextBox^ textBoxFrames;
    System::Windows::Forms::Label^ labelPages;
    System::Windows::Forms::TextBox^ textBoxPages;
    System::Windows::Forms::Label^ labelRefString;
    System::Windows::Forms::TextBox^ textBoxRefString;
    System::Windows::Forms::Button^ buttonSubmit;
    System::Windows::Forms::Label^ labelStatus;
    System::Windows::Forms::Label^ labelHeader;
    System::ComponentModel::Container^ components;

private:
    // InitializeComponent method
    void InitializeComponent(void)
    {
        this->labelHeader = (gcnew
System::Windows::Forms::Label());
        this->labelFrames = (gcnew
System::Windows::Forms::Label());
        this->textBoxFrames = (gcnew
System::Windows::Forms::TextBox());
        this->labelPages = (gcnew
System::Windows::Forms::Label());
        this->textBoxPages = (gcnew
System::Windows::Forms::TextBox());
        this->labelRefString = (gcnew
System::Windows::Forms::Label());
        this->textBoxRefString = (gcnew
System::Windows::Forms::TextBox());
        this->buttonSubmit = (gcnew
System::Windows::Forms::Button());
        this->labelStatus = (gcnew
System::Windows::Forms::Label());
        this->SuspendLayout();
        //
        // labelHeader

```

```

//
this->labelHeader->AutoSize = true;
this->labelHeader->Font = (gcnew
System::Drawing::Font(L"Segoe UI", 18,
System::Drawing::FontStyle::Bold));
this->labelHeader->ForeColor =
System::Drawing::Color::FromArgb(static_cast<System::Int32>(
static_cast<System::Byte>(102)),
static_cast<System::Int32>(static_cast<System::Byte>(51)),
static_cast<System::Int32>(static_cast<System::Byte>(153)));
this->labelHeader->Location =
System::Drawing::Point(116, 28);
this->labelHeader->Name = L"labelHeader";
this->labelHeader->Size = System::Drawing::Size(216,
32);
this->labelHeader->TabIndex = 0;
this->labelHeader->Text = L"SMART SWAPPER";
this->labelHeader->TextAlign =
System::Drawing::ContentAlignment::MiddleCenter;
//
// labelFrames
//
this->labelFrames->AutoSize = true;
this->labelFrames->Font = (gcnew
System::Drawing::Font(L"Microsoft Sans Serif", 12));
this->labelFrames->Location =
System::Drawing::Point(49, 86);
this->labelFrames->Name = L"labelFrames";
this->labelFrames->Size = System::Drawing::Size(67,
20);
this->labelFrames->TabIndex = 1;
this->labelFrames->Text = L"Frames:";
//
// textBoxFrames
//
this->textBoxFrames->BackColor =
System::Drawing::Color::White;
this->textBoxFrames->Font = (gcnew
System::Drawing::Font(L"Microsoft Sans Serif", 9.75F));
this->textBoxFrames->Location =
System::Drawing::Point(206, 84);
this->textBoxFrames->Name = L"textBoxFrames";

```

```
this->textBoxFrames->Size = System::Drawing::Size(150,
22);
this->textBoxFrames->TabIndex = 2;
//
// labelPages
//
this->labelPages->AutoSize = true;
this->labelPages->Font = (gcnew
System::Drawing::Font(L"Microsoft Sans Serif", 12));
this->labelPages->Location = System::Drawing::Point(49,
135);
this->labelPages->Name = L"labelPages";
this->labelPages->Size = System::Drawing::Size(58, 20);
this->labelPages->TabIndex = 3;
this->labelPages->Text = L"Pages:";
//
// textBoxPages
//
this->textBoxPages->BackColor =
System::Drawing::Color::White;
this->textBoxPages->Font = (gcnew
System::Drawing::Font(L"Microsoft Sans Serif", 9.75F));
this->textBoxPages->Location =
System::Drawing::Point(206, 135);
this->textBoxPages->Name = L"textBoxPages";
this->textBoxPages->Size = System::Drawing::Size(150,
22);
this->textBoxPages->TabIndex = 4;
//
// labelRefString
//
this->labelRefString->AutoSize = true;
this->labelRefString->Font = (gcnew
System::Drawing::Font(L"Microsoft Sans Serif", 12));
this->labelRefString->Location =
System::Drawing::Point(49, 182);
this->labelRefString->Name = L"labelRefString";
this->labelRefString->Size = System::Drawing::Size(134,
20);
this->labelRefString->TabIndex = 5;
this->labelRefString->Text = L"Reference String:";
//
// textBoxRefString
```

```

//
this->textBoxRefString->BackColor =
System::Drawing::Color::White;
this->textBoxRefString->Font = (gcnew
System::Drawing::Font(L"Microsoft Sans Serif", 9.75F));
this->textBoxRefString->Location =
System::Drawing::Point(206, 182);
this->textBoxRefString->Name = L"textBoxRefString";
this->textBoxRefString->Size =
System::Drawing::Size(200, 22);
this->textBoxRefString->TabIndex = 6;
//
// buttonSubmit
//
this->buttonSubmit->BackColor =
System::Drawing::Color::FromArgb(static_cast<System::Int32>(
static_cast<System::Byte>(153)),
static_cast<System::Int32>(static_cast<System::Byte>(102)),
static_cast<System::Int32>(static_cast<System::Byte>(204)));
this->buttonSubmit->FlatStyle =
System::Windows::Forms::FlatStyle::Flat;
this->buttonSubmit->Font = (gcnew
System::Drawing::Font(L"Microsoft Sans Serif", 9.75F,
System::Drawing::FontStyle::Bold));
this->buttonSubmit->ForeColor =
System::Drawing::Color::White;
this->buttonSubmit->Location =
System::Drawing::Point(150, 256);
this->buttonSubmit->Name = L"buttonSubmit";
this->buttonSubmit->Size = System::Drawing::Size(150,
35);
this->buttonSubmit->TabIndex = 7;
this->buttonSubmit->Text = L"SUBMIT";
this->buttonSubmit->UseVisualStyleBackColor = false;
this->buttonSubmit->Click += gcnew
System::EventHandler(this, &Form1::buttonSubmit_Click);
//
// labelStatus
//
this->labelStatus->AutoSize = true;
this->labelStatus->Font = (gcnew
System::Drawing::Font(L"Microsoft Sans Serif", 9.75F));

```

```

        this->labelStatus->Location = System::Drawing::Point(50,
240);
        this->labelStatus->Name = L"labelStatus";
        this->labelStatus->Size = System::Drawing::Size(0, 16);
        this->labelStatus->TabIndex = 8;
        //
        // Form1
        //
        this->AutoScaleDimensions = System::Drawing::SizeF(6,
13);
        this->AutoScaleMode =
System::Windows::Forms::AutoScaleMode::Font;
        this->BackColor =
System::Drawing::Color::FromArgb(static_cast<System::Int32>(
static_cast<System::Byte>(230)),
static_cast<System::Int32>(static_cast<System::Byte>(230)),
static_cast<System::Int32>(static_cast<System::Byte>(250)));
        this->ClientSize = System::Drawing::Size(458, 346);
        this->Controls->Add(this->labelStatus);
        this->Controls->Add(this->buttonSubmit);
        this->Controls->Add(this->textBoxRefString);
        this->Controls->Add(this->labelRefString);
        this->Controls->Add(this->textBoxPages);
        this->Controls->Add(this->labelPages);
        this->Controls->Add(this->textBoxFrames);
        this->Controls->Add(this->labelFrames);
        this->Controls->Add(this->labelHeader);
        this->Name = L"Form1";
        this->Text = L"Smart Swapper - Page Replacement";
        this->ResumeLayout(false);
        this->PerformLayout();

    }

private:
    System::Void buttonSubmit_Click(System::Object^ sender,
System::EventArgs^ e)
    {
        int frames, pages;
        labelStatus->ForeColor = System::Drawing::Color::Red;

        try {

```

```

frames = Int32::Parse(textBoxFrames->Text);
pages = Int32::Parse(textBoxPages->Text);
System::String^ rawRef = textBoxRefString->Text;

// Split the reference string
cli::array<System::String^>^ rawArray = rawRef-
>Split(' ');

// Validate and store reference string
System::Collections::Generic::List<int>^ refList =
gcnew System::Collections::Generic::List<int>();
for each (System::String ^ s in rawArray)
{
    if (!System::String::IsNullOrWhiteSpace(s))
    {
        try {
            int val = Int32::Parse(s);
            refList->Add(val);
        }
        catch (...) {
            labelStatus->Text = "Reference string contains
invalid numbers!";
            return;
        }
    }
}

if (frames <= 0 || pages <= 0) {
    labelStatus->Text = "Frames and pages must be
positive integers.";
    return;
}
if (refList->Count != pages) {
    labelStatus->Text = "Reference string must contain
exactly " + pages + " numbers.";
    return;
}

labelStatus->ForeColor =
System::Drawing::Color::FromArgb(static_cast<System::Int32>(
static_cast<System::Byte>(0)),

```

```
static_cast<System::Int32>(static_cast<System::Byte>(128)),
static_cast<System::Int32>(static_cast<System::Byte>(0)));
    labelStatus->Text = "Input accepted!";

    // Open Form2 and pass data
    Form2^ f2 = gcnew Form2(frames, pages, refList);
    f2->Show();
    this->Hide();
}
catch (System::Exception^ ex) {
    labelStatus->Text = "Invalid input! " + ex->Message;
}
}
};
}
```

Form1.cpp

```
#include "Form1.h"

using namespace::System;

using namespace::System::Windows::Forms;

void Main(cli::array<System::String^>^ args) {

    Application::EnableVisualStyles();

    Application::SetCompatibleTextRenderingDefault(false);

    Project1::Form1 form;

    Application::Run(% form);

}
```