📋 Homework-8: Implementation of FIFO and Second Chance Page Replacement Algorithms

> ⚠️ **Important:** This homework focuses on implementing two page replacement algorithms: First-In, First-Out (FIFO) and Second Chance (Clock Algorithm). Students will work with memory management concepts including page frames, page faults, reference bits, and circular queue structures.
>
> 💡 All the necessary concepts required to complete this homework are covered in **Module 5: Memory Management and textbook chapter on memory management.**
>
> **You must use these strict gcc compilation flags while building your application:**
> **-std=c17**
> **-D_POSIX_C_SOURCE=200809L**
> **-Wall**
> **-Wextra**
> **-Werror**
> **-g**
> **-O0**
>
> ⚠️ **There is only one submission - RESUBMISSION IS NOT ALLOWED. You must test your implementation with the provided testing framework on the Ocelot server before the final submission.**

📧 Grader Contact Information

For any questions or concerns related to assignment grading, please reach out to the TA listed below.

Communication Guidelines:

- Ensure that all communication is polite and respectful
- **You must contact the Graders first for any grading issues**
- If the matter remains unresolved, feel free to reach out to the instructor

Name: Instructor – bbhatkal@fiu.edu (⚠️ via the Canvas inbox only)
Section: COP 4610-U01 (84957)

Name: Angie Martinez – amart1070@fiu.edu (⚠️ via the Canvas inbox only)
Section: COP 4610-U02 (85031)

Name: Srushti Visweswaraiah – svisw003@fiu.edu(⚠️ via the Canvas inbox only)
Sections: COP 4610-UHA (85236)

☑ Section 1: Homework Overview

To implement two fundamental page replacement algorithms: **FIFO (First-In, First-Out) and Second Chance (Clock Algorithm)** that manage page frames in virtual memory systems, demonstrating how operating systems handle page faults and optimize memory utilization.

☑ Section 2: Learning Objectives

- Understand and implement the FIFO page replacement algorithm.
- Understand and implement the Second Chance (Clock) page replacement algorithm.
- Work with page frames and manage page fault occurrences.
- Implement reference bit management for the Second Chance algorithm.
- Handle circular queue structures for victim frame selection.
- Calculate page fault rates and hit ratios.
- Compare the performance characteristics of different page replacement strategies.

☑ Section 3: Problem Description

You will implement two **page replacement algorithms** that operating systems use to manage limited physical memory when demand exceeds available page frames.

Algorithm 1: FIFO (First-In, First-Out)

The **FIFO algorithm** replaces the oldest page in memory (the one that has been in memory the longest).

Algorithm Overview:

1. **Frame Management**: Pages are loaded into frames in sequential order
2. **Replacement Policy**: Replace the oldest page (first-in) when all frames are full
3. **Circular Queue**: Use a circular queue structure with a victim pointer
4. **Page Hit**: If requested page is already in memory, no replacement occurs
5. **Page Fault**: If requested page is not in memory, replace the oldest page

Key Characteristics:

- Simple to implement and understand
- Uses a circular FIFO queue
- Can suffer from Belady's Anomaly (more frames may cause more page faults)
- Does not consider page usage patterns

Input Format

Your program will read from a file named `INPUT.txt` with the following format:

```
Reference string = 1,2,3,4,1,2,5,1,2,3,4,5
Number of frame = 3
```

Input Specifications:

- First line contains the page reference string (comma-separated integers)
- Second line contains the number of available frames
- Page numbers are non-negative integers
- Frame count is a positive integer

Output Format

Your output must exactly match this format:

```
FIFO Page Replacement Algorithm
--------------------------------
Page 7 loaded. Page fault occurred. Frames: [ 7  _  _ ]
Page 0 loaded. Page fault occurred. Frames: [ 7  0  _ ]
Page 1 loaded. Page fault occurred. Frames: [ 7  0  1 ]
Page 2 loaded. Page fault occurred. Frames: [ 2  0  1 ]
Page 0 already in memory. No page fault. Frames: [ 2  0  1 ]
Page 3 loaded. Page fault occurred. Frames: [ 2  3  1 ]
Page 0 loaded. Page fault occurred. Frames: [ 2  3  0 ]
Page 2 already in memory. No page fault. Frames: [ 2  3  0 ]
Page 0 already in memory. No page fault. Frames: [ 2  3  0 ]
Page 4 loaded. Page fault occurred. Frames: [ 4  3  0 ]
Page 0 already in memory. No page fault. Frames: [ 4  3  0 ]
Page 2 loaded. Page fault occurred. Frames: [ 4  2  0 ]

FIFO Summary
==============
Reference String: 7 0 1 2 0 3 0 2 0 4 0 2
Page Faults:      Y Y Y Y N Y Y N N Y N Y
Total Page Faults: 8
Hit Ratio (Number_of_Page_Hits / Total_Number_of_Pages): 33.33%
```

Algorithm 2: Second Chance (Clock Algorithm)

The **Second Chance algorithm** is an improvement over FIFO that considers whether a page has been recently referenced before replacing it.

Algorithm Overview:

1. **Reference Bit**: Each frame has a reference bit (0 or 1)
2. **On Page Load**: When a page is loaded, its reference bit is set to 1
3. **On Page Hit**: When a page is accessed, its reference bit is set to 1
4. **Victim Selection**:
    - Start from current victim pointer position
    - If reference bit = 0, replace the page
    - If reference bit = 1, give it a "second chance" by clearing the bit to 0 and move to next frame
    - Continue in circular fashion until a page with reference bit = 0 is found
5. **Circular Queue**: Victim pointer wraps around to frame 0 after reaching the last frame
6. **Victim Page Pointer (Clock Hand) Advancement**: Victim Page Pointer (Clock Hand) advances each time it clears a reference bit to 0, and also after replacing a page (***not on page hits***)

Key Characteristics:

- Prevents recently used pages from being replaced
- Uses a circular queue with reference bits
- Industry standard used in real operating systems

Input Format

Your program will read from a file named `INPUT.txt` with the following format:

```
Reference string = 1,2,3,4,1,2,5,1,2,3,4,5
Number of frame = 3
```

Input Specifications:

- First line contains the page reference string (comma-separated integers)
- Second line contains the number of available frames
- Page numbers are non-negative integers
- Frame count is a positive integer

Output Format

Your output must exactly match this format:

```
Second Chance (Clock) Page Replacement Algorithm
-----------------------------------------------
Page 7 loaded into empty frame. Frames: [ 7  _  _ ]
Page 0 loaded into empty frame. Frames: [ 7  0  _ ]
Page 1 loaded into empty frame. Frames: [ 7  0  1 ]
Page 2 replaces page with reference bit 0. Frames: [ 2  0  1 ]
Page 0 already in memory. Reference bit set. Frames: [ 2  0  1 ]
Page 3 replaces page with reference bit 0. Frames: [ 2  0  3 ]
Page 0 already in memory. Reference bit set. Frames: [ 2  0  3 ]
Page 2 already in memory. Reference bit set. Frames: [ 2  0  3 ]
```

```
Page 0 already in memory. Reference bit set. Frames: [ 2  0  3 ]
Page 4 replaces page with reference bit 0. Frames: [ 4  0  3 ]
Page 0 already in memory. Reference bit set. Frames: [ 4  0  3 ]
Page 2 replaces page with reference bit 0. Frames: [ 4  0  2 ]

Second Chance Summary
=====================
Reference String: 7 0 1 2 0 3 0 2 0 4 0 2
Page Faults:      Y Y Y Y N Y N N N Y N Y
Total Page Faults: 7
Hit Ratio (Number_of_Page_Hits / Total_Number_of_Pages): 41.67%
```

Output Requirements:

- Show step-by-step execution for each page reference
- Display frame contents after each page reference
- Use `_` for empty frames
- Show "loaded" for page faults, "already in memory" for hits
- Display summary with reference string, fault indicators (Y/N), total faults, and hit ratio
- **Exact spacing and formatting required** - frames shown as `[ 1  0  7 ]` with single spaces

Key Concepts

Page Fault: Occurs when a requested page is not in any frame

- Requires loading the page from disk (simulated)
- Triggers replacement algorithm if all frames are full

Page Hit: Occurs when a requested page is already in a frame

- No replacement needed
- For Second Chance: reference bit is set to 1

Hit Ratio: Percentage of page references that result in hits

- Formula: `(Total References - Page Faults) / Total References × 100`
- Higher hit ratio indicates better performance

Reference Bit (for Second Chance Algorithm only):

- Set to 1 when page is loaded or accessed
- Cleared to 0 when page is given a second chance
- Used to determine victim for replacement

Victim Page Pointer, also called the *Clock Hand* (Second Chance only):

- Initially set to frame 0
- Advances each time it clears a reference bit to 0, and also after replacing a page (***not on page hits***)
- Wraps around circularly (after last frame, goes to frame 0)

☑ Section 4: Provided Files Framework

📖 **Refer to these files** for understanding the exact requirements.

⚠️ **Please do not make any modifications to the framework files provided.** You are required to implement **only** the file `submission_HW8.c`. Any changes to the framework files may result in errors during evaluation or loss of credit.

Folder Structure

```
PROVIDED_FILES/
├── skeleton_submission_HW8.c  # Implementation template file (rename to submission_HW8.c)
├── page_replacement.h         # Header file (Do not modify!)
├── driver.c                   # Main driver program (Do not modify!)
├── Makefile                   # Build system (Do not modify!)
├── autograder_HW8.sh          # Testing script (Do not modify!)
├── batchgrader_HW8.sh         # Batch testing script (Do not modify!)
├── Sample_Executable/
    └── page_replacement       # Sample executable for testing
└── Testing/                   # Test cases (Do not modify!)
    ├── Testcases/             # Simple, Moderatem Rigorous testcases
        ├── simpme.txt
        ├── moderate.txt
        ├── rigorous.txt
        └── Expected_Output/   # Corresponding Expected Outputs
        ├── fifo_simple_output.txt
        ├── fifo_moderate_output.txt
        ├── fifo_rigorous_output.txt
        ├── second_chance_simple_output.txt
        ├── second_chance_moderate_output.txt
        └── second_chance_rigorous_output
```

☑ Section 5: Submission Requirements

File Requirements (⚠️ exact names required)

- **submission_HW8.c**: Your complete implementation containing both FIFO and Second Chance algorithms
- **README.txt or README.pdf or README.md**: Student information and documentation
    - Your README ⚠️ **MUST include**:

```
# Student Information
- Full Name: [Your Full Name]
- PID: [Your FIU Panther ID]
- Section: [Your Course Section]
- FIU Email: [Your @fiu.edu email]

# Homework: FIFO and Second Chance Page Replacement
[Brief description of your implementation approach]
[Explain how you implemented the FIFO algorithm]
[Explain how you implemented the Second Chance algorithm]
[Describe how you handled the circular queue and reference bits]
[Mention any challenges faced and how you solved them]
```

Deliverable Folder (`ZIP file`) Structure - ⚠️ exact structure required - no additional files/subfolders

> ⚠️ **DO NOT submit any other files other than these required files. The batch autograder may treat any additional items in the ZIP file as invalid, which will result in a grade of zero: submission_HW8.c, README.txt (or README.pdf or README.md)**
>
> 📁 All other required framework files will be supplied by the instructor to the autograder during final grading.

- ⚠️ You are required to submit your work as **a single ZIP archive file**.
- ⚠️ **Filename Format**: your `Firstname_Lastname.zip` (**exact format required**).
- **You will be held responsible for receiving a ZERO grade if the submission guidelines are not followed**.

```
Harry_Potter.zip
├── submission_HW8.c  # Your FIFO and Second Chance implementation
└── README.txt        # Your details and implementation approach
```

☑️ Section 6: Developing and Testing Your Implementation

Development Workflow

RECOMMENDED DEVELOPMENT ORDER:

1. **Start with FIFO** - simpler algorithm, builds foundation
2. **Test FIFO thoroughly** before moving to Second Chance
3. **Implement Second Chance** - builds on FIFO with reference bits
4. **Test Second Chance** with various scenarios
5. **Validate output format** matches exactly with expected output

STEP 1 - Sample Executable Testing:

```
# Test FIFO implementation
./page_replacement Testing/Testcases/simple.txt fifo > my_fifo_output.txt

# Test Second Chance implementation
./page_replacement Testing/Testcases/simple.txt second_chance > my_second_chance_output.txt
```

Study the output carefully to understand:

- How frames are displayed: `[ 1  0  7 ]`
- When page faults vs hits occur
- How the summary section is formatted
- Exact text for each message

STEP 2 - Manual testing:

Ensure the following files are located in the **same directory**:

- **Your implementation**: `submission_HW8.c`
- **All the provided framework files**
- **The provided testcase folder:** `Testing/`

```
# Explore the Makefile first to understand build targets and compilation settings
cat Makefile

# Build the executable
make rebuild

# Test your FIFO implementation
./page_replacement Testing/Testcases/simple.txt fifo > my_fifo_output.txt
diff my_fifo_output.txt Testing/Expected_Output/fifo_simple_output.txt

# Test your Second Chance implementation
./page_replacement Testing/Testcases/simple.txt second_chance > my_second_chance_output.txt
diff my_second_chance_output.txt Testing/Expected_Output/second_chance_simple_output.txt

# If diff shows no output, your implementation is correct!
# Repeat the above steps for Moderate and Rigorous testcases
```

STEP 3 - Autograder testing:

Ensure the following files are located in the **same directory**, and then run the **Autograder**:

- **Your implementation**: `submission_HW8.c`
- **All the provided framework files**
- **The provided testcase folder:** `Testing/`

```
# Make autograder executable
chmod +x autograder_HW8.sh

# Run the autograder
./autograder_HW8.sh
```

- **Check your grade** and fix any issues
- **Repeat until** you achieve the desired score

STEP 4 - Batch Autograder testing:

> ⚠️ **Important:** The batch autograder processes multiple submissions. Please be patient!

The batch autograder, **used by the instructor**, processes all student submissions at once. It utilizes the provided framework files, extracts the required files from your submitted `.zip` file, and performs grading accordingly.

⚠️ **Backup Files Before Running Batch Grader - The script deletes existing files to rebuild the environment from scratch.**

Ensure the following files are located in the **same directory**, and then run the **Batch Autograder**:

- **All the provided framework files**
- **The provided testcase folder:** `Testing/`
- **Your final submission** `ZIP` **file** consisting the required files for the submission: example, `Harry_Potter.zip`

```
# Make batch autograder executable
chmod +x batchgrader_HW8.sh

# Run the batch-autograder
./batchgrader_HW8.sh
```

Final Testing requirements:

- ⚠️ **MUST test on ocelot server**: `ocelot-bbhatkal.aul.fiu.edu`
- ⚠️ **Test thoroughly before submission** - no excuses accepted
- ⚠️ **"Works on my computer" is NOT accepted** as an excuse
- ✅ **If you pass all test cases on the server, you will likely pass instructor's final grading**
- ✅ **What you see is what you get** - autograder results predict your final grade

---

☑️ Section 7: Grading Criteria

🤖 Autograder Testing

- ⚠️ Your implementation `submission_HW8.c` will be tested against the provided test case as well as additional instructor test cases using the autograder.
- ⚠️ **Exact output matching required** - any deviation results in point deduction
- ⚠️ **Program must compile** without errors or warnings

Test Case Distribution:

FIFO Algorithm (45 points total):

- **Simple Test Case (15 points)**: Basic FIFO operation with sequential page references
- **Moderate Test Case (15 points)**: FIFO with mixed hit/fault patterns
- **Rigorous Test Case (15points)**: Complex scenarios including edge cases (all frames full, repeated references, worst-case page faults)

Second Chance Algorithm (45 points total):

- **Simple Test Case (15 points)**: Basic Second Chance operation with reference bit management
- **Moderate Test Case (15 points)**: Reference bit clearing and victim pointer advancement
- **Rigorous Test Case (15 points)**: Complex scenarios with circular queue wrap-around, multiple second chances, and edge cases

Code Quality Manual Grading(10 points):

- Clear code structure and organization
- Meaningful variable names and comments
- Proper memory management
- Well-written README with implementation approach

⛔ Penalties

- ⚠️ **Missing README**: **ZERO grade** (batchgrader compilation failure)
- ⚠️ **Missing** `submission_HW8.c` : **ZERO grade** (autograder compilation failure)
- ⚠️ **Incorrect ZIP filename**: **ZERO grade** (autograder extraction failure)
- ⚠️ **Wrong source filename**: **ZERO grade** (autograder compilation failure)
- 🔒 **Resubmission**: **NOT ALLOWED**

☑ Section 8: Technical Specifications

Required Function Signatures

Your `submission_HW8.c` must implement these exact function signatures:

```c
void fifo(int pages[], int n, int frames);
void second_chance(int pages[], int n, int frames);
```

Function Parameters:

- `pages[]`: Array containing the page reference string
- `n`: Number of page references in the string
- `frames`: Number of available page frames

Data Structures

You may use these data structures:

```c
int frame[MAX_FRAMES];           // Page frames array
int reference[MAX_FRAMES];       // Reference bits for Second Chance (0 or 1)
int k;                           // Victim frame pointer
int page_faults;                 // Counter for page faults
char page_fault_occurrence[];    // Track Y/N for each reference
```

☑ Section 9: Academic Integrity

- 👤 **This is an individual assignment**
- 💬 **You may discuss concepts** but not share code
- 📝 **All submitted code** must be your original work
- 📚 **You are encouraged to refer to textbook and lecture materials** but must write your own implementation
- ⚠️ **Plagiarism** is a serious offense and will result in penalties - **ZERO grade** (academic integrity violation).

🚀 **Good luck with your FIFO and Second Chance Page Replacement implementation!**