1. 개요

- 페이지 교체 알고리즘 구현 및 폴트 횟수 측정

- 페이지 교체 알고리즘 종류
(1) Optimal
(2) FIFO
(3) LIFO
(4) LRU (Least Recently Used)
(5) LFU (Least Frequently Used)
(6) SC (Second Chance / One handed Clock)
(7) ESC (Enhanced Second Chance / Two handed Clock / NUR(Not Recently Used))

- 페이지 교체 알고리즘 동작 방식, Pseudo code, 소스 코드
(1) Optimal
* 동작 방식:
미래에 가장 오랫동안 사용되지 않을 페이지 교체
1. 페이지프레임에 검사할 스트링 번호가 존재하는지 검사 - 존재: Hit, 존재하지 않음: Fault
2. Fault가 날 경우 - 빈 프레임이 있을 경우: 빈 프레임에 해당 스트링 번호 입력
3. Fault가 날 경우 - 빈 프레임이 없을 경우:
   미래의 스트링에서 가장 오랫동안 사용되지 않는 페이지를 스트링 번호로 교체
   미래에 사용되지 않는 프레임이 2개 이상이면 가장 먼저 들어온 페이지 교체(FIFO)

* Pseudo code:

```
1. 입력: 페이지프레임 수-num, 페이지스트링-page_str
2. fault, 2차원 배열 arr, 배열 queue 0으로 초기화
3. for k=0...PAGELEN do
4.       exist 0으로 초기화
5.       for i=0...num
6.               if arr[i][0]==page_str[k] then
7.                       exist = 1
8.                       break
9.               end if
10.              arr[i][1] = 0
11.      end for
12.      if exist==0 do
13.              done 0으로 초기화
14.              for i=0...num do
15.                      if arr[i][0]==0 then
16.                              arr[i][0]에 page_str[k] 입력
17.                              queue[i]에 page_str[k] 입력
18.                              done = 1
19.                              break
20.                      end if
21.              end for
22.              if done==0 then
23.                      count 1로 초기화
24.                      for i=k...PAGELEN do
```

```
25.                             if count<=num then
26.                                     for j=0...num do
27.                                             if arr[j][0]==page_str[i] and arr[j][1]==0 then
28.                                                     arr[j][1]값 count로 업데이트 후 count 1 증가
30.                                             end if
31.                                     end for
32.                             end if
33.                             else then break
35.                     end for
36.                     for i=0...num do
37.                             done_2 0으로 초기화
38.                             for j=0...num do
39.                                     if queue[i]==arr[j][0] and (arr[j][1]==0 or arr[j][1]>=num) then
40.                                             arr[j][0]값 page_str[k]로 교체
41.                                             for l=i...(num-1) do queue[i]=queue[i+1]
42.                                             queue[num-1]에 page_str[k] 삽입
43.                                             done_2=1
44.                                             break
45.                                     end if
46.                             end for
47.                             if done_2==1 then break
48.                     end for
49.             end for
50.             fault 1 증가
51.     end if
52. end for
```

*참고 문헌:
강의 자료
www.cs.utexas.edu/users/witchel/372/lectures/16.PageReplacementAlgos.pdf p.3

(2) FIFO
* 동작 방식:
가장 먼저 들어온 페이지 교체
1. 페이지프레임에 검사할 스트링 번호가 존재하는지 검사 - 존재: Hit, 존재하지 않음: Fault
2. Fault가 날 경우 - 빈 프레임이 있을 경우: 빈 프레임에 해당 스트링 번호 입력
3. Fault가 날 경우 - 빈 프레임이 없을 경우: 가장 먼저 들어온 페이지 교체

* Pseudo code:

```
1. 입력: 페이지프레임 수-num, 페이지스트링-page_str
2. fault, idx, 배열 arr 0으로 초기화
3. for k=0...PAGELEN do
4.      exist 0으로 초기화
5.      for i=0...num
6.              if arr[i]==page_str[k] then
7.                      exist = 1
8.                      break
9.              end if
10.     end for
11.     if exist==0 then
```

```
12.             done 0으로 초기화
13.             if arr[idx%num]==0 then
14.                     arr[idx%num]에 page_str[k] 입력
15.                     done = 1
16.             end if
17.             if done==0 then arr[idx%num]값 page_str[k]으로 교체
18.             fault 1 증가
19.             idx 1 증가
20.     end if
21. end for
```

*참고 문헌:
강의 자료
www.cs.utexas.edu/users/witchel/372/lectures/16.PageReplacementAlgos.pdf p4

(3) LIFO
* 동작 방식:
가장 나중에 들어온 페이지 교체
1. 페이지프레임에 검사할 스트링 번호가 존재하는지 검사 - 존재: Hit, 존재하지 않음: Fault
2. Fault가 날 경우 - 빈 프레임이 있을 경우: 빈 프레임에 해당 스트링 번호 입력
3. Fault가 날 경우 - 빈 프레임이 없을 경우: 가장 나중에 들어온 프레임 교체

* Pseudo code:

```
1. 입력: 페이지프레임 수-num, 페이지스트링-page_str
2. fault, 배열 arr 0으로 초기화
3. for k=0...PAGELEN do
4.      exist 0으로 초기화
5.      for i=0...num
6.              if arr[i]==page_str[k] then
7.                      exist = 1
8.                      break
9.              end if
10.     end for
11.     if exist==0 then
12.             done 0으로 초기화
13.             for i=0...num do
14.                     if arr[i]==0 then
15,                             arr[i][0]에 page_str[k] 입력
16,                             done = 1
17,                             break
18.                     end if
19.             end for
20.             if done==0 then arr[num-1]를 page_str[k]로 교체
21.             fault 1 증가
22.     end if
23. end for
```

*참고 문헌:
강의 자료
www.scaler.com/topics/operating-system/page-replacement-algorithm/

(4) LRU

* 동작 방식:

가장 오랫동안 참조되지 않은 페이지 교체

1. 페이지프레임에 검사할 스트링 번호가 존재하는지 검사
   - 존재: Hit(마지막 참조 시점 업데이트), 존재하지 않음: Fault
2. Fault가 날 경우 - 빈 프레임이 있을 경우: 빈 프레임에 해당 스트링 번호 입력, 참조 시간 초기화
3. Fault가 날 경우 - 빈 프레임이 없을 경우: 가장 오랫동안 참조되지 않은 페이지 교체, 참조 시간 초기화

* Pseudo code:

```
1. 입력: 페이지프레임 수-num, 페이지스트링-page_str
2. fault, 2차원 배열 arr, 0으로 초기화, time -1로 초기화
3. for k=0...PAGELEN do
4.       exist 0으로 초기화, time 1 증가
5.       for i=0...num
6.               if arr[i][0]==page_str[k] then
7.                       arr[i][1]값 time으로 업데이트
8.                       exist = 1
9.                       break
10.              end if
11.      end for
12.      if exist==0 then
13.              done 0으로 초기화
14.              for i=0...num do
15.                      if arr[i][0]==0 then
16,                              arr[i][0]에 page_str[k] 입력
17.                              arr[i][1]에 time 입력
18.                              done = 1
19.                              break
20.                      end if
21.              end for
22.              if done==0 then
23.                      idx 0으로 초기화
24.                      for i=1...num do
25.                              if arr[idx][1]>arr[i][1] then idx=i
26.                      end for
27.                      arr[idx][0] page_str[k]로 교체
28.                      arr[idx][1] time으로 교체
29.              end if
30.              fault 1 증가
31.      end if
32. end for
```

*참고 문헌:

강의 자료

www.cs.utexas.edu/users/witchel/372/lectures/16.PageReplacementAlgos.pdf p5

(5) LFU

* 동작 방식:

가장 참조 횟수가 적은 페이지 교체

1. 페이지프레임에 검사할 스트링 번호가 존재하는지 검사
   - 존재: Hit(참조 횟수 업데이트), 존재하지 않음: Fault
2. Fault가 날 경우 - 빈 프레임이 있을 경우: 빈 프레임에 해당 스트링 번호 입력, 참조 횟수 초기화
3. Fault가 날 경우 - 빈 프레임이 없을 경우:
   가장 참조 횟수가 적은 페이지 교체, 참조 횟수 초기화
   가장 참조 횟수가 적은 페이지가 2개 이상이면 그 중 먼저 들어온 페이지 교체(FIFO)

* Pseudo code:

```
1. 입력: 페이지프레임 수-num, 페이지스트링-page_str
2. fault, 2차원 배열 arr, 배열 queue 0으로 초기화
3. for k=0...PAGELEN do
4.      exist 0으로 초기화
5.      for i=0...num
6.              if arr[i][0]==page_str[k] then
7.                      참조 횟수 arr[i][1] 1 증가
8.                      exist = 1
9.                      break
10.             end if
11.     end for
12.     if exist==0 then
13.             done 0으로 초기화
14.             for i=0...num do
15.                     if arr[i][0]==0 then
16,                             arr[i][0]에 page_str[k] 입력
17.                             arr[i][1] 0으로 초기화
18.                             queue[i]에 page_str[k] 입력
19.                             done = 1
20.                             break
21.                     end if
22.             end for
23.             if done==0 then
24.                     idx 0으로 초기화
25.                     for i=1...num do
26.                             for j=1...num do
27.                                     if i==0 and queue[i]==arr[j][0] then idx에 j 입력
28.                                     else if queue[i]==arr[j][0] and arr[idx][1]>arr[j][1] then
29.                                             idx j로 업데이트
30.                                     end else if
31.                             end for
32.                     end for
33.                     arr[idx][0] page_str[k]로 교체
34.                     arr[idx][1] 0으로 초기화
35.                     for i=idx...(num-1) then queue[i]=queue[i+1]
36.             end if
37.             fault 1 증가
38.     end if
39. end for
```

*참고 문헌:
강의 자료

(6) SC

* 동작 방식:

1. 페이지프레임에 검사할 스트링 번호가 존재하는지 검사
   - 존재: Hit(R비트 1로 업데이트), 존재하지 않음: Fault
2. Fault가 날 경우 - 빈 프레임이 있을 경우: 빈 프레임에 해당 스트링 번호 입력, R비트 1로 업데이트
3. Fault가 날 경우 - 빈 프레임이 없을 경우:
   교체가 진행될 때까지 순회
      R비트가 0일 경우: 해당 페이지 교체 R비트 1로 초기화
      R비트가 1일 경우: 해당 페이지 R비트 0으로 교체

* Pseudo code:

```
1. 입력: 페이지프레임 수-num, 페이지스트링-page_str
2. fault, idx, 2차원 배열 arr 0으로 초기화
3. for k=0...PAGELEN do
4.       exist 0으로 초기화
5.       for i=0...num
6.               if arr[i][0]==page_str[k] then
7.                       R비트 arr[i][1]에 1 입력
8.                       exist = 1
9.                       break
10.              end if
11.      end for
12.      if exist==0 then
13.              done 0으로 초기화
14.              if arr[idx%num][0]==0 then
15,                      arr[idx%num][0]에 page_str[k] 입력
16.                      arr[idx%num][1] 1로 초기화
17.                      idx 1 증가
18.                      done = 1
19.              end if
20.              if done==0 then
21.                      while 1 do
22.                              if arr[idx%num][1]==0 then
23.                                      arr[idx%num][0] page_str[k]로 교체
24.                                      arr[idx%num][1] 1로 초기화
25.                                      idx 1 증가
26.                                      break
27.                              end if
28.                              else then arr[idx%num][1] 0으로 업데이트
29.                      end while
30.              end if
31.              fault 1 증가
32.      end if
33. end for
```

*참고 문헌:

강의 자료
https://www.cs.utexas.edu/users/witchel/372/lectures/16.PageReplacementAlgos.pdf 7,8

(7) ESC

* 동작 방식:

1. 페이지프레임에 검사할 스트링 번호가 존재하는지 검사
   - 존재: Hit(R일 때 RD비트 10, W일 때 RD비트 11로 업데이트), 존재하지 않음: Fault
2. Fault가 날 경우 - 빈 프레임이 있을 경우:
   빈 프레임에 해당 스트링 번호 입력, R일 때 RD비트 10, W일 때 RD비트 11로 입력
3. Fault가 날 경우 - 빈 프레임이 없을 경우:
   교체가 진행될 때까지 순회
   RD비트가 00일 경우: 해당 페이지 교체, R일 때 RD비트 10, W일 때 RD비트 11로 교체
   RD비트가 10 or 01일 경우: 해당 페이지 RD비트 00으로 교체
   RD비트가 11일 경우: 해당 페이지 RD비트 01으로 교체

* Pseudo code:

```
1. 입력: 페이지프레임 수-num, 페이지스트링-page_str, 페이지스트링 RW비트-page_rw
2. fault, idx, 2차원 배열 arr 0으로 초기화
3. for k=0...PAGELEN do
4.       exist 0으로 초기화
5.       for i=0...num
6.               if arr[i][0]==page_str[k] then
7.                       R비트 arr[i][1]에 1 입력
8.                       if page_rw[k]==0 then arr[i][2]에 0 입력
9.                       else then arr[i][2]에 1 입력
10.                      exist=1
11.                      break
12.              end if
13.      end for
14.      if exist==0 then
15.              done 0으로 초기화
16.              if arr[idx%num][0]==0 then
17,                      arr[idx%num][0]에 page_str[k] 입력
18.                      arr[idx%num][1] 1로 초기화
19.                      if page_rw[k]==0 then arr[i][2]에 0 입력
20.                      else then arr[i][2]에 1 입력
21.                      idx 1 증가
22.                      done = 1
23.              end if
24.              if done==0 then
25.                      while 1 do
26.                              if arr[idx%num][1]==0 and arr[idx%num][2]==0 then
27.                                      arr[idx%num][0] page_str[k]로 교체
28.                                      arr[idx%num][1] 1로 초기화
29                                       if page_rw[k]==0 then arr[i][2] 0으로 교체
30.                                      else then arr[i][2] 1로 교체
31                                       idx 1 증가
32.                                      break
33.                              end if
34.                              else if (arr[idx%num][1]==0 and arr[idx%num][2]==1) or
                                         (arr[idx%num][1]==1 and arr[idx%num][2]==0) then
35.                                      arr[idx%num][1] 0으로 업데이트
36.                                      arr[idx%num][2] 0으로 업데이트
```

```
37.                                 end else if
38.                                 else if arr[idx%num][1]==1 and arr[idx%num][2]==1 then
39.                                         arr[idx%num][1] 0으로 업데이트
40.                                         arr[idx%num][2] 1으로 업데이트
41.                                 end else if
42.                                 idx 1 증가
43.                         end while
44.                 end if
45.                 fault 1 증가
46.         end if
47. end for
```

*참고 문헌:

www.cs.utexas.edu/users/witchel/372/lectures/16.PageReplacementAlgos.pdf 7,8

2. 소스 코드

(1) main.c - Page Replacement Algorithm Simulator 실행

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include "algorithm.h"

#define BUFSIZE 1024    //입력 버퍼 사이즈
#define PAGELEN 500 //페이지 스트링 길이

static int page_str[PAGELEN];   //페이지 스트링
static int page_rw[PAGELEN];    //페이지 스트링 RW비트

void menu_c(int sel);   //C.데이터 입력 방식 선택 구현 함수
int main(){
    printf("Page Replacement Algorithm Simulator\n\n"); //프로그램 이름 출력

    //A.페이지 교체 알고리즘 선택 메뉴
    printf("A. Page Replacement 알고리즘 시뮬레이터를 선택하시오.(최대 3개)\n");    //메뉴 설명. 최대 3개임을 명시
    printf("(1)Optimal (2)FIFO (3)LIFO (4)LRU (5)LFU (6)SC (7)ESC (8)ALL\n");   //총 8개의 경우의 수

    int num_a[4];           //최대 3개의 선택한 숫자를 저장, 배열의 네번째는 4개 이상의 입력이 들어왔는지 확인을 위한 자리
    while(1){           //올바른 알고리즘이 선택될 때까지 반복하여 재입력
        for(int i=0; i<4; i++)  //저장 배열 -1로 초기화
            num_a[i] = -1;
        int a = 0;      //선택한 수를 넣을 배열의 인덱스를 가리키는 변수
        char buf[BUFSIZE];  //입력 버퍼
        int err = 0;            //에러 여부를 가리키는 변수
        // 선택하는 알고리즘 입력, string형으로 받아 입력 버퍼에 저장
        printf("Input: ");
        fgets(buf, BUFSIZE, stdin);
        //버퍼에 저장한 입력을 띄어쓰기, 콤마, 탭, 띄어쓰기를 기준으로 구분한 첫번째 입력을 token에 저장.
        char *token;
        token = strtok(buf, " ,\t\n");
        while(token != NULL && a < 4){          //버퍼가 완전히 비워지고, a가 3이하일 때까지 반복
            int temp = atoi(token);         //token 안의 입력을 정수형으로 바꿔 temp에 저장
            // 오류 출력 후 재입력
            // temp = 0 : 0, int형을 변환할 수 없는 값
            // temp = 1 : 1
            // temp < 1 or temp > 7 : 1~8 사이가 아닌 값
            if(temp == 0 || temp == -1 || (temp < 1 || temp > 8)){
                printf("Wrong Input. 1~7 사이의 숫자를 선택하시오.\n");
                err = 1;    //에러가 일어남을 저장
                break;      //재입력을 받기 위해 while문 break
            }
            num_a[a++] = temp;          //에러가 나지 않았을 경우 num_a에 선택된 숫자를 저장하고, 선택된 수를 넣을 인덱스를 하나 올림
            token = strtok(NULL, " ,\t\n"); //버퍼의 다음 입력을 구분하여 token에 저장
        }
        //에러 여부 검사, 에러가 났으면 재입력
        if(err==1)
            continue;
        //8이 다른 수와 함께 입력 됐을 경우 재입력
        //첫 번째가 8이고 두 번째에 입력이 저장됐을 경우
        //두 번째, 세 번째가 8인 경우
        if((num_a[0]==8 && num_a[1]!=-1) || num_a[1]==8 || num_a[2]==8){
            printf("Wrong Input. 8은 다른 숫자와 선택할 수 없습니다.\n");
            continue;
        }
        //입력이 없는 경우 재입력
        if(num_a[0] == -1){
            printf("Wrong Input. 최소 한 개의 숫자를 선택하시오.\n");
            continue;
        }
        //3개 이상 입력한 경우 재입력, 3개 이상 입력되었으면 배열의 네 번째의 값이 바뀜
        else if(num_a[3] != -1){
            printf("Wrong Input. 최대 세 개의 숫자를 선택하시오.\n");
            continue;
        }
        break;
    }
    //B.페이지 프레임 개수 입력
    int b, num_b = 0;   //b:scanf의 리턴값 저장, num_b:scanf로 정수를 입력받음
    printf("\nB. 페이지 프레임의 개수를 입력하시오.(3~10)\n");  //메뉴 설명, 3부터 10사이의 값이 들어와야 함을 명시
    while(1){   //올바른 프레임 개수가 선택될 때까지 반복하여 재입력
        //num_b에 입력받은 정보를 저장. scanf는 입력이 원하는 방식(정수)이면 1을 리턴
        printf("Input: ");
        b = scanf("%d", &num_b);
        while(getchar()!='\n'); //입력 버퍼 비움
        if(b == 1){ //입력받은 정보가 정수이면 실행
            if(num_b>= 3 && num_b<=10)  //입력받은 정수가 3이상 10이하이면 while문 탈출
                break;
            else                //입력받은 정수가 3미만 10초과이면 에러 출력 후 재입력
                printf("Wrong Input. 3~10 사이의 숫자를 선택하시오.\n");
        }       //입력받은 정보가 정수가 아니면 에러 출력 후 재입력
        else
            printf("Wrong Input. 3~10 사이의 숫자를 선택하시오.\n");
    }
    //C.데이터 입력 방식 선택
    int c, num_c = 0;   //c:scanf의 리턴값 저장, num_c:scanf로 정수를 입력받음
    printf("\nC. 데이터의 입력 방식을 선택하시오.(1,2)\n"); //메뉴 설명, 1또는 2가 입력되어야 함을 명시
    printf("(1)랜덤하게 생성\n(2)사용자 생성 파일 오픈\n");
    while(1){   //올바른 데이터 입력 방식이 선택될 때까지 반복하여 재입력
        //num_c에 입력받은 정보를 저장. scanf는 입력이 원하는 방식(정수)이면 1을 리턴
        printf("Input: ");
        c = scanf("%d", &num_c);
        while(getchar()!='\n'); //입력 버퍼 비움
        if(c == 1){ //입력받은 정보가 정수이면 실행
            if(num_c==1 || num_c==2)    //입력받은 정수가 1또는 2이면 while문 탈출
                break;
            else                //입력받은 정수가 1또는 2가 아니면 에러 출력 후 재입력
                printf("Wrong Input. 1 또는 2를 선택하시오. \n");
        }
        else    //입력받은 정보가 정수가 아니면 에러 출력 후 재입력
            printf("Wrong Input. Enter 1 또는 2를 선택하시오.\n");
    }

    menu_c(num_c);      //참조 페이지 스트링 생성. 선택된 입력 방식으로 데이터 생성 혹은 파일 오픈
    //시뮬레이션 결과를 저장할 파일 open
    FILE *fp;
    fp = fopen("save.txt", "w");

    for(int i=0; i<3; i++){ //num_a를 인덱스 0~2를 돌며 선택한 알고리즘 실행
        if(num_a[i]==-1)                //-1: 더 이상 선택된 알고리즘이 없음
            break;
        else if(num_a[i]==1)                //1: Optimal 알고리즘 수행
            opt(num_b, page_str, fp);
        else if(num_a[i]==2)                //2: FIFO 알고리즘 수행
            fifo(num_b, page_str, fp);
        else if(num_a[i]==3)                //3: LIFO 알고리즘 수행
            lifo(num_b, page_str, fp);
        else if(num_a[i]==4)                //4: LRU 알고리즘 수행
            lru(num_b, page_str, fp);
        else if(num_a[i]==5)                //5: LFU 알고리즘 수행
            lfu(num_b, page_str, fp);
        else if(num_a[i]==6)                //6: SC 알고리즘 수행
            sc(num_b, page_str, fp);
        else if(num_a[i]==7)                //7: ESC 알고리즘 수행
            esc(num_b, page_str, page_rw, fp);
        else if(num_a[i]==8){               //모든 알고리즘 수행
            opt(num_b, page_str, fp);
            fifo(num_b, page_str, fp);
            lifo(num_b, page_str, fp);
            lru(num_b, page_str, fp);
            lfu(num_b, page_str, fp);
            sc(num_b, page_str, fp);
            esc(num_b, page_str, page_rw, fp);
        }
    }
    fclose(fp); //시뮬레이션 결과를 저장한 파일 close

    printf("D. 종료.\n");   //종료
    return 0;
}
```

```c
//c. 데이터 입력 방식 구현
void menu_c(int sel){
    //랜덤 생성 선택시
    if(sel == 1){
        printf("Ramdomly generate.\n"); //선택한 입력 방식 출력
        srand(time(NULL));          //난수 생성
        for(int i=0; i<PAGELEN; i++){       //페이지 스트링 길이만큼 저장
            page_str[i] = rand()%30+1;  //1~30 정수 랜덤하게 저장
            page_rw[i] = rand()%2;      //0,1 랜덤하게 저장, 0:R, 1:W
        }
    }
    //사용자 생성 파일 오픈 선택시
    else if(sel == 2){
        FILE *fp;       //생성 파일 포인터
        char file_name[32]; //오픈할 파일 이름 변수

        printf("Open file. Input file name.");  //선택한 입력 방식 출력
        do{
            //파일 이름 입력
            printf("\nInput: ");
            scanf("%s", file_name);
        }while((fp = fopen(file_name, "r")) == NULL);   //입력받은 파일이 정상적으로 오픈될 때까지 반복하여 파일 이름 재입력, open

        printf("\nOpen %s\n\n", file_name); //입력한 파일 오픈 메시지
        int temp;   //입력 스트링 번호 버퍼
        char temp_2;    //입력 스트링 RW 버퍼
        for(int i=0; i<PAGELEN; i++){       //입력 스트링 개수만큼 저장
            fscanf(fp, "%d(%c) ", &temp, &temp_2);  //입력 스트링 번호와 RW여부를 read하여 버퍼에 저장
            page_str[i] = temp;                 //버퍼의 입력 스트링 번호 저장
            if(temp_2=='r')             //버퍼가 r을 가리키면 RW비트에 0 저장
                page_rw[i]=0;
            else                        //버퍼가 w를 가리키면 RW비트에 1 저장
                page_rw[i]=1;
        }
        fclose(fp); //생성 파일 close

    }
}
```

(2) algorithm.h - 알고리즘 구현 함수 선언

```c
#include <stdio.h>

void opt(int num, int *page_str, FILE *fp);             //optimal 알고리즘 함수 선언
void fifo(int num, int *page_str, FILE *fp);            //FIFO 알고리즘 함수 선언
void lifo(int num, int *page_str, FILE *fp);            //LIFO 알고리즘 함수 선언
void lru(int num, int *page_str, FILE *fp);             //LRU 알고리즘 함수 선언
void lfu(int num, int *page_str, FILE *fp);             //LFU 알고리즘 함수 선언
void sc(int num, int *page_str, FILE *fp);              //SC 알고리즘 함수 선언
void esc(int num, int *page_str, int *page_rw, FILE *fp);   //ESC 알고리즘 함수 선언
```

(3) algorithm.c - 알고리즘 구현 함수 정의

```c
#include <stdio.h>
#include <stdlib.h>
#include "algorithm.h"

//페이지 스트링 길이
#define PAGELEN 500
```

- opt

```c
//Optimal 함수 정의
void opt(int num, int *page_str, FILE* fp){ //인자 num:페이지프레임 개수, page_str:페이지스트링, fp:출력 저장 파일 포인터
    printf("Optimal\n");        //Optimal 메시지 출력
    fprintf(fp, "Optimal\n");   //Optimal 메시지 저장

    int fault = 0;          //fault 수 저장 0:페이지폴트 1:히트
    int **arr = (int**)calloc(num, sizeof(int*));           //페이지프레임, num만큼 동적으로 생성, 모든 변수 0으로 초기화
    for(int i=0; i<num; i++)
        arr[i] = (int*)calloc(2, sizeof(int));      //2차원 배열로 구현하여 arr[i][0]에 번호, arr[i][1]에 우선순위 저장
    int *arr_queue = (int*)calloc(num, sizeof(int));        //두 번째 정렬 기준으로 FIFO 사용을 위한 큐

    //전체 페이지스트링 검사
    for(int k=0; k<PAGELEN; k++){
        int exist = 0;      //fault여부 체크
        for(int i=0; i<num; i++){   //페이지프레임 검사
            if(arr[i][0]==page_str[k]){ //페이지프레임에 검사하는 스트링 번호가 존재할 경우
                exist = 1;      //fault가 나지 않음
                break;          //검사 종료
            }
            arr[i][1] = 0;          //존재하지 않을 경우 모든 우선순위 초기화
        }

        if(exist == 0){ //fault가 날 경우
            int done = 0;   //페이지프레임의 빈 자리 여부
            for(int i=0; i<num; i++){   //페이지프레임검사
                if(arr[i][0]==0){   //빈 자리가 있을 경우
                    arr[i][0] = page_str[k];    //빈 자리를 해당 스트링으로 교체
                    arr_queue[i] = page_str[k]; //큐에 스트링 저장하여 순서 기억
                    done = 1;               //빈자리가 있다는 것을 체크
                    break;                  //검사 종료
                }
            }
```

```c
            if(done == 0){  //fault가 나고 빈자리가 없을 경우
                int count = 1;  //각 프레임의 번호에 해당하는 미래의 페이지 스트링을 찾으면 1씩 증가
                for(int i=k; i<PAGELEN; i++){   //미래의 페이지스트링 검사
                    if(count <= num){   //모든 프레임의 번호을 찾을 때까지 탐색
                        for(int j=0; j<num; j++){   //모든 페이지프레임 검사
                            if(arr[j][0] == page_str[i] && arr[j][1] == 0)  //현재 페이지프레임과 번호가 같고, 현재 페이지 프레임을 아직 찾은 적이 없는 경우
                                arr[j][1] = count++;                //해당 프레임에 우선순위 저장
                        }
                    }
                    else            //모든 프레임을 찾았을 경우 종료
                        break;
                }   //모든 페이지스트링을 검사
                for(int i=0; i<num; i++){   //큐의 index
                    int done_2 = 0; //교체 여부 확인
                    for(int j=0; j<num; j++){   //모든 프레임 우선순위 저장
                        //큐에 저장된 FIFO를 지키며 탐색, 먼저 들어온 프레임을 먼저 검사하기 때문에, 같은 우선순위(미래에 존재하지 않는) 프레임이 2개 이상일 경우 FIFO에 따라 처리
                        if(arr_queue[i]==arr[j][0] && (arr[j][1]==0 || arr[j][1]>=num)){    //미래에서 번호를 찾지 못했거나, 가장 마지막에 나오는 경우
                            arr[j][0] = page_str[k];        //해당 프레임 교체
                            //큐 업데이트
                            for(int l=i; l<num-1; l++)
                                arr_queue[l] = arr_queue[l+1];
                            arr_queue[num-1] = page_str[k];
                            done_2 = 1; //교체 확인
                            break;      //교체가 이루어졌으면 검사 종료
                        }
                    }
                    if(done_2==1)   //교체가 이루어졌으면 검사 종료
                        break;
                }
            }
            fault++;    //페이지 폴트 수 1 증가
        }
```

```
        printf("%d: ", page_str[k]);        //검사할 페이지스트링 번호 출력
        fprintf(fp, "%d:    ", page_str[k]);    //검사할 페이지스트링 번호 저장
        //페이지프레임 번호 출력 및 저장
        for(int i=0; i<num; i++){
            if(arr[i][0] != 0){ //프레임이 비어 있지 않은 경우
                printf("|%d ", arr[i][0]);
                fprintf(fp, "|%d    ", arr[i][0]);
            }
            else{           //프레임이 비어 있는 경우
                printf("|   ");
                fprintf(fp, "|  ");
            }
        }
        if(exist == 0){ //페이지폴트가 이뤄진 경우
            printf("|F");       //폴트 출력
            fprintf(fp, "|F");  //폴트 저장
        }
        printf("\n");
        fprintf(fp, "\n");
    }
    printf("OPT Page Fault: %d\n\n", fault);        //전체 페이지폴트 수 출력
    fprintf(fp, "OPT Page Fault: %d\n\n", fault);   //전체 페이지폴트 수 저장

    //동적 할당 해제
    for(int i=0; i<num; i++)
        free(arr[i]);
    free(arr);
    free(arr_queue);
}
```

- fifo

```
//FIFO 함수 정의
void fifo(int num, int *page_str, FILE *fp){    //인자 num:페이지프레임 개수, page_str:페이지스트링, fp:출력 저장 파일 포인터
    printf("FIFO\n");       //FIFO 메시지 출력
    fprintf(fp, "FIFO\n");      //FIFO 메시지 저장

    int fault = 0;          //fault 수 저장
    int idx = 0;            //큐 구현을 위한 인덱스, 교체한 페이지의 수

    int *arr = (int*)calloc(num, sizeof(int));  //페이지프레임, num만큼 동적으로 생성, 모든 변수 0으로 초기화

    //전체 페이지스트링 검사
    for(int k=0; k<PAGELEN; k++){
        int exist = 0;      //fault여부 체크 0:페이지폴트 1:히트
        for(int i=0; i<num; i++){   //페이지프레임 검사
            if(arr[i]==page_str[k]){    //페이지프레임에 검사하는 스트링 번호가 존재할 경우
                exist = 1;      //히트
                break;          //검사 종료
            }
        }

        //FIFO는 페이지의 교체가 차례대로 이루어지기 때문에, 배열을 원형 큐로 구현, idx를 num으로 나눠 교체할 배열의 인덱스를 가리킴.
        if(exist == 0){ //fautt가 날 경우
            int done = 0;   //페이지프레임의 빈 자리 여부
            if(arr[idx%num]==0){    //빈 자리가 있을 경우
                arr[idx%num] = page_str[k]; //빈 자리를 해당 스트링으로 교체
                done = 1;           //빈자리가 있다는 것을 체크
            }
            if(done == 0)   //fault가 나고 빈자리가 없을 경우
                arr[idx%num] = page_str[k]; //가장 오래된 페이지 교체
            fault++;    //페이지 폴트 수 1 증가
            idx++;      //교체한 index 1 증가
        }
        printf("%d: ", page_str[k]);        //검사할 페이지스트링 번호 출력
        fprintf(fp, "%d:    ", page_str[k]);    //검사할 페이지스트링 번호 저장
        //페이지프레임 번호 출력 및 저장
        for(int i=0; i<num; i++){
            if(arr[i] != 0){    //프레임이 비어 있지 않은 경우
                printf("|%d ", arr[i]);
                fprintf(fp, "|%d    ", arr[l]);
            }
            else{           //프레임이 비어 있는 경우
                printf("|   ");
                fprintf(fp, "|  ");
            }
        }
        if(exist==0){   //페이지폴트가 이뤄진 경우
            printf("|F");       //폴트 출력
            fprintf(fp, "|F");  //폴터 저장
        }
        printf("\n");
        fprintf(fp, "\n");
    }
    printf("FIFO Page Fault: %d\n\n", fault);       //전체 페이지폴트 수 출력
    fprintf(fp, "FIFO Page Fault: %d\n\n", fault);  //전체 페이지폴트 수 저장

    free(arr);  //동적 할당 해제
}
```

- lifo

```
//LIFO 함수 정의
void lifo(int num, int *page_str, FILE *fp){    //인자 num:페이지프레임 개수, page_str:페이지스트링, fp:출력 저장 파일 포인터
    printf("LIFO\n");   //LIFO 메시지 출력
    fprintf(fp, "LIFO\n");  //LIFO 메시지 저장

    int fault = 0;      //fault 수 저장
    int *arr = (int*)calloc(num, sizeof(int));  //페이지프레임, num만큼 동적으로 생성, 모든 변수 0으로 초기화

    //전체 페이지스트링 검사
    for(int k=0; k<PAGELEN; k++){
        int exist = 0;  //fault 여부 체크
        for(int i=0; i<num; i++){   //페이지프레임 검사
            if(arr[i]==page_str[k]){    //페이지프레임에 검사하는 스트링 번호가 존재할 경우
                exist = 1;      //fault가 나지 않음
                break;          //검사 종료
            }
        }

        if(exist == 0){ //fault가 날 경우
            int done = 0;   //페이지프레임의 빈 자리 여부
            for(int i=0; i<num; i++){   //페이지프레임 검사
                if(arr[i]==0){  //빈 자리가 있을 경우
                    arr[i] = page_str[k];   //빈 자리를 해당 스트링으로 교체
                    done = 1;       //빈자리가 있다는 것을 체크
                    break;          //검사 종료
                }
            }
            if(done == 0)       //fault가 나고 빈자리가 없을 경우
                arr[num-1] = page_str[k];   //마지막에 교체된 페이지 교체
            fault++;    //페이지 폴트 수 1 증가
        }
        printf("%d: ", page_str[k]);        //검사할 페이지스트링 번호 출력
        fprintf(fp, "%d:    ", page_str[k]);    //검사할 페이지스트링 번호 저장
        //페이지프레임 번호 출력 및 저장
        for(int i=0; i<num; i++){
            if(arr[i] != 0){    //프레임이 비어 있지 않은 경우
                printf("|%d ", arr[i]);
                fprintf(fp, "|%d    ", arr[i]);
            }
            else{           //프레임이 비어 있는 경우
                printf("|   ");
                fprintf(fp, "|  ");
            }
        }
        if(exist==0){   //페이지폴트가 이뤄진 경우
            printf("|F");       //폴트 출력
            fprintf(fp, "|F");  //폴트 저장
        }
        printf("\n");
        fprintf(fp, "\n");
    }
    printf("LIFO Page Fault: %d\n\n", fault);       //전체 페이지폴트 수 출력
    fprintf(fp, "LIFO Page Fault: %d\n\n", fault);  //전체 페이지폴트 수 저장

    //동적 할당 해제
    free(arr);
}
```

- lru

```c
//LRU 함수 정의
void lru(int num, int *page_str, FILE *fp){ //인자 num:페이지프레임 개수, page_str:페이지스트링, fp: 출력 저장 파일 포인터
    printf("LRU(Least Recent Used)\n");      //LRU 메시지 출력
    fprintf(fp, "LRU(Least Recent Used)\n");    //LRU 메시지 저장

    int fault = 0;  //fault 수 저장
    int **arr = (int**)calloc(num, sizeof(int*));   //페이지프레임, num만큼 동적으로 생성, 모든 변수 0으로 초기화
    for(int i=0; i<num; i++)
        arr[i] = (int*)calloc(2, sizeof(int));  //2차원 배열로 구현하여 arr[i][0]에 번호, arr[i][1]에 최근에 사용한 시점 저장

    int time = -1;  //현재 시점

    //전체 페이지 스트링 검사
    for(int k=0; k<PAGELEN; k++){
        time++; //시점 변화, 1 증가
        int exist = 0;  //fault 여부 체크 0:페이지폴트 1:히트
        for(int i=0; i<num; i++){   //페이지 프레임 검사
            if(arr[i][0]==page_str[k]){ //페이지프레임에검사하는 스트링 번호가 존재할 경우
                arr[i][1] = time;   //최근에 사용한 시점 업데이트
                exist = 1;      //히트
                break;          //검사 종료
            }
        }

        if(exist == 0){ //fault가 날 경우
            int done = 0;   //페이지 프레임의 빈 자리 여부
            for(int i=0; i<num; i++){   //페이지프레임검사
                if(arr[i][0]==0){   //빈 자리가 있을 경우
                    arr[i][0] = page_str[k];    //빈 자리를 해당 스트링으로 교체
                    arr[i][1] = time;       //최근 사용한 시점 업데이트
                    done = 1;           // 빈자리가 있다는 것을 체크
                    break;              //검사종료
                }
            }

            if(done == 0){  //fault가 나고 빈자리가 없을 경우
                int idx = 0;    //가장 사용한 시점이 빠른 인덱스 저장
                for(int i=1; i< num; i++){  // 페이지 검사
                    if(arr[idx][1] > arr[i][1]) // 만약 해당 인덱스의 시점이 더 빠른 경우
                        idx = i;        // 인덱스 업데이트
                }
                arr[idx][0] = page_str[k];  //해당 인덱스의 페이지 교체
                arr[idx][1] = time;     //해당 인덱스의 페이지 사용 시점 업데이트
            }
            fault++;    // 페이지 폴트 수 1 증가
        }
        printf("%d: ", page_str[k]);        //검사할 페이지스트링 번호 출력
        fprintf(fp, "%d:    ", page_str[k]);    // 검사할 페이지스트링 번호 저장
        //페이지프레임 번호 출력 및 저장
        for(int i=0; i<num; i++){
            if(arr[i][0] != 0){ //프레임이 비어 있지 않은 경우
                printf("|%d ", arr[i][0]);
                fprintf(fp, "|%d    ", arr[i][0]);
            }
            else{           //프레임이 비어 있는 경우
                printf("|   ");
                fprintf(fp, "|  ");
            }
        }
        if(exist==0){   //페이지폴트가 이뤄진 경우
            printf("|F");     //폴트 출력
            fprintf(fp, "|F");  //폴트 저장
        }
        printf("\n");
        fprintf(fp, "\n");
    }
    printf("LRU Page Fault: %d\n\n", fault);        //전체 페이지폴트 수 출력
    fprintf(fp, "LRU Page Fault: %d\n\n", fault);   //전체 페이지폴트 수 저장
    //동적 할당 해제
    for(int i=0; i<num; i++)
        free(arr[i]);
    free(arr);
}
```

- lfu

```c
//LFU 함수 정의
void lfu(int num, int *page_str, FILE *fp){ //인자 num:페이지프레임 개수, page_str:페이지스트링, fp:출력 저장 파일 포인터
    printf("LFU (Least Frequently Used)\n");    //LFU 메시지 출력
    fprintf(fp, "LFU (Least Frequently Used)\n");   //LFU 메시지 저장

    int fault = 0;  //fault 수 저장
    int **arr = (int**)calloc(num, sizeof(int*));       //페이지프레임, num만큼 동적으로 생성, 모든 변수 0으로 초기화
    for(int i=0; i<num; i++)
        arr[i] = (int*)calloc(2, sizeof(int));  //2차원 배열로 구현하여 arr[i][0]에 번호, arr[i][1]에 참조 횟수 저장
    int *arr_queue = (int*)calloc(num, sizeof(int));    //두 번째 정렬 기준으로 FIFO 사용을 위한 큐
    //전체 페이지스트링 검사
    for(int k=0; k<PAGELEN; k++){
        int exist = 0;  //fault여부 체크
        for(int i=0; i<num; i++){   //페이지프레임 검사
            if(arr[i][0]==page_str[k]){ //페이지프레임에 검사하는 스트링 번호가 존재할 경우
                arr[i][1] += 1; //참조 횟수 1 증가
                exist = 1;      //fault가 나지 않음
                break;          //검사 종료
            }
        }

        if(exist == 0){ //fault가 날 경우
            int done = 0;   //페이지 프레임의 빈 자리 여부
            for(int i=0; i<num; i++){   //페이지프레임 검사
                if(arr[i][0]==0){   //빈 자리가 있을 경우
                    arr[i][0] = page_str[k];    //빈 자리를 해당 스트링으로 교체
                    arr[i][1] = 0;          //참조 횟수수 0으로 초기화
                    arr_queue[i]=page_str[k];   //큐에 스트링 저장하여 순서 기억
                    done = 1;           //빈자리가 있다는 것을 체크
                    break;              //검사 종료
                }
            }

            if(done == 0){  //fault가 날 경우
                int idx = 0;    //가장 참조 횟수가 적은 페이지 인덱스 저장
                for(int i=0; i<num; i++){   //큐의 index
                    for(int j=1; j< num; j++){  //모든 프레임 검사
                        //큐를 사용하여 FIFO를 바탕으로 탐색, 먼저 들어온 프레임을 먼저 감사하기 때문에, 같은 참조 횟수를 가진 프레임을 FIFO에 따라 처리
                        if(i==0 && arr_queue[i]==arr[j][0]) //큐의 가장 먼저온 프레임의 인덱스로 횟수가 가장 적은 프레임 초기화
                            idx = j;
                        else if(arr_queue[i]==arr[j][0] && arr[idx][1] > arr[j][1]) //해당 인덱스의 프레임의 참조 횟수가 적으면 업데이트
                            idx = j;
                    }
                }

                arr[idx][0] = page_str[k];  //참조 횟수가 가장 적은 프레임 교체
                arr[idx][1] = 0;        //참조 횟수 초기화
                //큐 업데이트
                for(int i=idx; i<num-1; i++)
                    arr_queue[i] = arr_queue[i+1];
                arr_queue[num-1] = page_str[k];
            }
            fault++;    //페이지 폴트 수 1 증가
        }
```

```
343         printf("%d: ", page_str[k]);        //검사할 페이지스트링 번호 출력
344         fprintf(fp, "%d:    ", page_str[k]);    //검사할 페이지스트링 번호 저장
345         //페이지프레임 번호 출력 및 저장
346         for(int i=0; i<num; i++){
347             if(arr[i][0] != 0){ //프레임이 비어 있지 않은 경우
348                 printf("|%d ", arr[i][0]);
349                 fprintf(fp, "|%d    ", arr[i][0]);
350             }
351             else{           //프레임이 비어 있는 경우
352                 printf("|   ");
353                 fprintf(fp, "|  ");
354             }
355         }
356         if(exist==0){   //페이지폴트가 이뤄진 경우
357             printf("|F");       //폴트 출력
358             fprintf(fp, "|F");  //폴트 저장
359         }
360         printf("\n");
361         fprintf(fp, "\n");
362     }
363     printf("LFU Page Fault: %d\n\n", fault);        //전체 페이지폴트 수 출력
364     fprintf(fp, "LFU Page Fault: %d\n\n", fault);   //전체 페이지폴트 수 저장
365     //동적 할당 해제
366     for(int i=0; i<num; i++)
367         free(arr[i]);
368     free(arr);
369     free(arr_queue);
370 }
```

- sc

```
371 //SC 함수 정의
372 void sc(int num, int *page_str, FILE *fp){  //인자 num:페이지프레임 개수, page_str:페이지스트링, fp:출력 저장 파일 포인터
373     printf("SC (Second Chance / One handed Clock)\n");      //SC 메시지 출력
374     fprintf(fp, "SC (Second Chance / One handed Clock)\n"); //SC 메시지 저장
375
376     int fault = 0;  //fault 수 저장
377     int idx = 0;    //검사할 인덱스, 원형 큐 구현을 위한 인덱스, 교체한 페이지의 수
378     int **arr = (int**)calloc(num, sizeof(int*));           //페이지프레임, num만큼 동적으로 생성, 모든 변수 0으로 초기화
379     for(int i=0; i<num; i++)
380         arr[i] = (int*)calloc(2, sizeof(int));  //2차원 배열로 구현하여 arr[i][0]에 번호, arr[i][1]에 Reference bit 저장
381     //전체 페이지스트링 검사
382     for(int k=0; k<PAGELEN; k++){
383         int exist = 0;  //fault 여부 체크
384         for(int i=0; i<num; i++){   //페이지프레임 검사
385             if(arr[i][0]==page_str[k]){ //페이지프레임에 검사하는 스트링 번호가 존재할 경우
386                 arr[i][1] = 1;      //Reference bit 1로 설정
387                 exist = 1;      //히트
388                 break;          //검사 종료
389             }
390         }
391         if(exist == 0){ // fault가 날 경우
392             int done = 0;   //페이지프레임 빈 자리 여부
393             if(arr[idx%num][0]==0){ //빈 자리가 있을 경우
394                 arr[idx%num][0] = page_str[k];  //빈 자리를 해당 스트링으로 교체
395                 arr[idx%num][1] = 1;            //Reference bit 1으로 초기화
396                 idx++;                  //검사할 큐 인덱스 1 증가
397                 done = 1;               //빈자리가 있다는 것을 체크
398             }
399             if(done==0){                        //fault가 나고 빈자리가 없을 경우
400                 while(1){               //교체가 진행될 때까지 순회
401                     if(arr[idx%num][1]==0){ //Reference bit가 0인 경우
402                         arr[idx%num][0] = page_str[k];  //해당 프레임 교체
403                         arr[idx%num][1] = 1;            //R 비트 초기화
404                         idx++;                  //검사할 큐 인덱스 1 증가
405                         break;                  //검사 탈출
406                     }
407                     else                //Referece bit가 1인 경우
408                         arr[idx%num][1] = 0;    //Referece bit 0으로 업데이트
409                     idx++;              //검사할 큐 인덱스 1 증가
410                 }
411             }
412             fault++;    //페이지 폴트 수 1 증가
413         }
```

```
414         printf("%d: ", page_str[k]);        //검사할 페이지스트링 번호 출력
415         fprintf(fp, "%d:    ", page_str[k]);    //검사할 페이지스트링 번호 저장
416         //페이지프레임 번호 출력 및 저장
417         for(int i=0; i<num; i++){
418             if(arr[i][0] != 0){ //프레임이 비어 있지 않은 경우
419                 printf("|%d", arr[i][0]);
420                 fprintf(fp, "|%d", arr[i][0]);
421                 if(arr[i][1]==1){   //R 비트가 1인 경우만 * 출력
422                     printf("*   ");
423                     fprintf(fp, "*  ");
424                 }
425                 else{           //R비트가 0인 경우
426                     printf("    ");
427                     fprintf(fp, "   ");
428                 }
429             }
430             else{           //프레임이 비어 있는 경우
431                 printf("|   ");
432                 fprintf(fp, "|  ");
433             }
434         }
435         if(exist==0){   //페이지 폴트가 이뤄진 경우
436             printf("|F");       //폴트 출력
437             fprintf(fp, "|F");  //폴트 저장
438         }
439         printf("\n");
440         fprintf(fp, "\n");
441     }
442     printf("SC Page Fault: %d\n\n", fault);     //전체 페이지 폴트 수 출력
443     fprintf(fp, "SC Page Fault: %d\n\n", fault);    //전체 페이지 폴트 수 저장
444
445     //동적 할당 해제
446     for(int i=0; i<num; i++)
447         free(arr[i]);
448     free(arr);
449 }
```

- esc

```
450 //ESC 함수 정의
451 void esc(int num, int *page_str, int *page_rw, FILE *fp){   //인자 num:페이지프레임 개수, page_str:페이지스트링, page_rw:페이지스트링 RW비트, fp:출력 저장 파일 포인터
452     printf("ESC (Enhanced Second Chance / Two handed clork)\n");       //ESC 메시지 출력
453     fprintf(fp, "ESC (Enhanced Second Chance / Two handed clork)\n");   //ESC 메시지 출력
454
455     int fault = 0;  //fault 수 저장
456     int idx = 0;    //검사할 인덱스, 원형 큐 구현을 위한 인덱스, 교체한 페이지의 수
457     int **arr = (int**)calloc(num, sizeof(int*));   //페이지프레임, num 만큼 동적으로 생성, 모든 변수 0으로 초기화
458     for(int i=0; i<num; i++)
459         arr[i] = (int*)calloc(3, sizeof(int));  //2차원 배열로 구현하여 arr[i][0]에 번호, arr[i][1]에 R비트, arr[i][2]에 D비트 저장
460     //전체 페이지스트링 검사
461     for(int k=0; k<PAGELEN; k++){
462         int exist = 0;  //fault 여부 체크
463         for(int i=0; i<num; i++){   //페이지 프레임 검사
464             if(arr[i][0]==page_str[k]){ //페이지프레임에 검사하는 스트링 번호가 존재할 경우
465                 arr[i][1] = 1;      //R비트 1로 업데이트
466                 if(page_rw[k]==0)   //Read일 경우
467                     arr[i][2] = 0;  //D비트 0으로 업데이트
468                 else            //Write일 경우
469                     arr[i][2] = 1;  //D비트 1로 업데이트
470                 exist = 1;
471                 break;
472             }
473         }
```

```
        if(exist == 0){ //fault가 날 경우
            int done = 0;   //페이지프레임의 빈 자리 여부
            if(arr[idx%num][0]==0){ //빈 자리가 있을 경우
                arr[idx%num][0] = page_str[k];  //빈 자리를 해당 스트링으로 교체
                arr[idx%num][1] = 1;            //R비트 1로 초기화
                if(page_rw[k]==0)           //Read일 경우
                    arr[idx%num][2] = 0;        //D비트 0으로 초기화
                else                        //Write일 경우
                    arr[idx%num][2] = 1;        //D비트 1로 초기화
                idx++;                  //검사할 인덱스 1 증가
                done = 1;               //빈 자리가 있다는 것을 체크
            }
            if(done==0){            //fault가 나고 빈자리가 없을 경우
                while(1){           //교체가 진행될 때까지 순회
                    if(arr[idx%num][1]==0 && arr[idx%num][2]==0){   //R bit: 0, D bot: 0
                        arr[idx%num][0] = page_str[k];  //해당 프레임 교체
                        arr[idx%num][1] = 1;            //R 비트 1로 업데이트
                        if(page_rw[k]==0)           //Read일 경우
                            arr[idx%num][2] = 0;        //D비트 0으로 업데이트
                        else                        //Write일 경우
                            arr[idx%num][2] = 1;        //D비트 1로 업데이트
                        idx++;                  //검사할 큐 인덱스 1 증가
                        break;                  //검사 종료
                    }
                    else if((arr[idx%num][1]==0 && arr[idx%num][2]==1) || (arr[idx%num][1]==1 && arr[idx%num][2]==0)){  //R bit: 0, D bot: 1 or R bit: 1, D bot: 0
                        arr[idx%num][1] = 0;    //R비트 1로 업데이트
                        arr[idx%num][2] = 0;    //D비트 0으로 업데이트
                    }
                    else if(arr[idx%num][1]==1 && arr[idx%num][2]==1){  //R bit: 1, D bot: 1
                        arr[idx%num][1] = 0;    //R비트 0로 업데이트
                        arr[idx%num][2] = 1;    //D비트 1으로 업데이트
                    }
                    idx++;              //검사할 큐 인덱스 1 증가
                }

            fault++;    //페이지 폴트 수 1증가
            }
        }
```

```
        //검사할 페이지스트링 번호 및 RW비트 출력 및 저장
        if(page_rw[k]==0){  //페이지스트링 Read
            printf("%d(R):  ", page_str[k]);
            fprintf(fp, "%d(R): ", page_str[k]);
        }
        else{           //페이지스트링 Write
            printf("%d(W):  ", page_str[k]);
            fprintf(fp, "%d(W): ", page_str[k]);
        }
        //페이지프레임 번호 출력 및 저장
        for(int i=0; i<num; i++){
            if(arr[i][0] != 0){ //프레임이 비어 있지 않은 경우
                printf("|%d(%d%d)   ", arr[i][0], arr[i][1], arr[i][2]);
                fprintf(fp, "| %d(%d%d) ", arr[i][0], arr[i][1], arr[i][2]);
            }
            else{           //프레임이 비어 있는 경우
                printf("|   ");
                fprintf(fp, "|      ");
            }
        }

        if(exist==0){   //페이지 폴트가 이뤄진 경우
            printf("|F");       //폴트 출력
            fprintf(fp, "|F");  //폴트 저장
        }
        printf("\n");
        fprintf(fp, "\n");
    }
    printf("ESC Page Fault: %d\n\n", fault);        //전체 페이지폴트 수 출력
    fprintf(fp, "ESC Page Fault: %d\n\n", fault);   //전체 페이지폴트 수 저장
    //동적 할당 해제
    for(int i=0; i<num; i++)
        free(arr[i]);
    free(arr);
}
```

(4) Makefile

```
CC=gcc              #컴파일러
CFLAGS=-g -Wall     #컴파일 옵션
TARGET=a.out            #빌드 대상 이름
OBJS=algorithm.o main.o #Object 파일 목록

#a.out 빌드
$(TARGET): $(OBJS)
    $(CC) -o $@ $(OBJS)

#algorithm.c 컴파일
algorithm.o: algorithm.c
    $(CC) -c -o algorithm.o algorithm.c

#main.c 컴파일
main.o: main.c
    $(CC) -c -o main.o main.c

#make clean 구현
clean:
    rm $(OBJECT) $(TARGET)
```

(5) make.c – ‘C.2 사용자 생성 파일 오픈’에서 사용할 파일 생성

- 따로 Makefile에 사용되지는 않으며, 따로 컴파일하여 실행해야 함.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>

#define PAGELEN 500

int main(){
    //페이지스트링 생성
    int str[PAGELEN];
    srand(time(NULL));                  //난수 설정
    for(int i=0; i<PAGELEN; i++)
        str[i] = rand()%30+1;           //1~30 랜덤으로 페이지스트링 번호 설정

    char st[PAGELEN][10];
    for(int i=0; i<PAGELEN; i++)
        sprintf(st[i], "%d", str[i]);   //페이지스트링 번호 스트링 형태로 변환

    //프레임스트링 RW
    int rw[PAGELEN];
    for(int i=0; i<PAGELEN; i++)
        rw[i] = rand()%2;               //0 or 1을 랜덤으로 설정. 0:Read 1:Write


    FILE *fp;                           //페이지스트링을 저장할 파일 포인터
    fp = fopen("string.txt", "w");      //string.txt로 새로운 파일 open
    for(int i=0; i<PAGELEN; i++){
        //페이지스트링 번호 쓰기
        fputs(st[i], fp);
        //페이지스트링 RW 쓰기
        fputs("(", fp);
        if(rw[i]==0)
            fputs("r", fp);
        else
            fputs("w", fp);
        fputs(")", fp);
        fputs(" ", fp);
    }
    fclose(fp);                         //파일 close
    return 0;
}
```

3. 사용자 매뉴얼

- 해당 코드는 디폴트로 페이지 스트링의 수를 500개로 설정하고 있으며, main.c, algorithm.c, make.c의 PAGELEN을 원하는 수로 변경하여 페이지스트링의 수를 원하는 수로 변경할 수 있다.

- a.out으로 실행한다.

(1) A. Page Repalcement 알고리즘 시뮬레이터를 선택하시오. (최대 3개)
- 초기 화면(선택이 끝나면 다음 메뉴로 넘어간다.)

- 1~7의 번호로 메뉴를 숫자로 입력하여 선택하며, 1~3 개가 선택 가능하다. 번호는 space로 구별한다. 그 외에 3개를 초과하여 입력하거나 잘못된 입력이 들어오면 에러를 보이고, 재입력 받는다.

- 8은 단독으로만 사용 가능하다.

(2) B. 페이지 프레임의 개수를 입력하시오. (3~10)
- 초기 화면(선택이 끝나면 다음 메뉴로 넘어간다.)

- 3~10 사이의 정수를 입력하고, 명시하지 않은 경우에 대해서 에러를 보이고 재입력 받는다.

(3) 데이터를 입력 방식을 선택하시오.
- 초기화면(선택이 끝나면 결과 출력으로 넘어간다.)

- 1 또는 2의 값을 받는다. 1은 랜덤하게 데이터를 생성하고, 2는 사용자 생성 파일 오픈 메뉴를 실행한다. 명시하지 않은 경우에 대해서 에러를 보이고 재입력 받는다.

- 2를 선택한 경우, 오픈할 파일 이름을 입력한다. 이 때, 올바르지 않은 파일 이름이 입력되었을 경우 재입력 받는다. ex) 예시로 string.txt 첨부

```
c. 데이터의 입력 방식을 선택하시오.(1,2)
(1)랜덤하게 생성
(2)사용자 생성 파일 오픈
Input: 2
Open file. Input file name.
Input: page2.txt

Input: page.txt

Open page.txt

Optimal
1:      |1      |       |       |F
2:      |1      |2      |       |F
3:      |1      |2      |3      |F
4:      |1      |4      |3      |F
```

- 오픈 파일의 예시는 다음과 같다.

```
열기(O)        string.txt        저장(S)
1 12(w) 7(w) 27(r) 12(r) 5(w) 18(r) 6(r) 2(r) 3(w) 10(r) 11(r) 19(r) 24(r) 9(w) 23(r) 28(r) 30(w) 6(w) 9(w) 9(r) 2(w) 11(w) 7(w) 26(r) 20(r) 19(r) 17(w) 24(w) 12(w) 2(r) 20(r) 15(r) 8(w) 9(r) 26(r) 5(r)
  26(r) 1(r) 28(w) 28(r) 10(r) 30(w) 9(r) 3(r) 30(r) 23(r) 23(w) 21(w) 20(w) 1(r) 21(w) 21(w) 3(r) 28(r) 9(w) 22(r) 8(w) 25(r) 7(w) 11(r) 26(r) 26(r) 25(r) 26(r) 4(r) 21(w) 22(r) 29(w) 13(w) 11(w) 19(w)
  15(r) 10(r) 19(w) 9(r) 1(r) 11(w) 1(w) 21(w) 22(w) 23(r) 11(r) 5(r) 17(r) 30(w) 13(w) 30(r) 7(r) 7(w) 6(r) 17(r) 25(r) 2(r) 4(r) 20(r) 27(r) 16(w) 3(r) 18(w) 28(w) 13(r) 6(r) 4(w) 22(r) 24(w) 13(w)
  22(w) 26(w) 5(w) 4(r) 9(r) 28(w) 14(w) 13(r) 14(w) 6(r) 25(w) 14(w) 12(r) 2(r) 11(w) 21(r) 18(r) 4(w) 24(r) 7(r) 1(r) 9(r) 9(w) 18(w) 28(w) 21(w) 15(r) 2(r) 4(w) 30(w) 6(w) 17(r) 17(r) 10(r) 20(w)
  25(r) 7(w) 25(r) 8(r) 13(r) 30(r) 2(w) 18(r) 4(r) 25(r) 20(w) 24(r) 12(r) 24(w) 9(r) 10(w) 24(w) 9(r) 10(r) 3(r) 6(w) 22(w) 17(r) 29(w) 25(r) 8(r) 4(w) 3(w) 24(r) 14(w) 14(r) 18(w) 12(w) 9(r) 25(w)
  16(w) 8(r) 19(w) 3(w) 11(r) 13(w) 23(w) 26(r) 17(w) 8(r) 4(r) 26(r) 23(w) 12(w) 28(r) 25(w) 10(r) 19(w) 3(r) 8(r) 6(r) 2(r) 12(w) 8(w) 25(r) 25(r) 22(w) 12(w) 28(w) 22(w) 29(w) 14(w) 21(w) 17(r) 16(r)
  2(w) 21(w) 30(w) 19(w) 29(r) 29(w) 23(w) 17(w) 21(r) 26(r) 14(w) 7(r) 5(w) 24(w) 9(r) 13(w) 29(w) 2(r) 16(r) 29(r) 26(r) 2(r) 20(w) 30(r) 29(r) 3(r) 20(w) 12(w) 23(w) 28(w) 20(w) 16(r) 18(r) 11(w) 5(w)
  9(r) 10(r) 27(r) 25(w) 30(w) 14(w) 30(w) 7(w) 19(r) 23(r) 7(w) 23(w) 22(w) 9(w) 8(w) 20(w) 4(w) 9(w) 1(r) 25(w) 7(w) 25(r) 14(r) 11(r) 17(w) 11(w) 30(r) 3(w) 21(w) 10(w) 7(r) 29(r) 19(w) 25(r) 15(w)
  11(r) 8(w) 14(r) 17(w) 26(w) 28(r) 15(r) 10(r) 11(w) 23(w) 17(w) 30(r) 19(w) 17(w) 22(r) 13(r) 16(r) 16(r) 19(w) 26(w) 3(w) 29(r) 17(w) 27(r) 11(r) 18(r) 3(r) 9(r) 29(w) 27(w) 15(w) 9(r) 26(r) 20(r)
  17(w) 22(r) 18(r) 1(r) 23(r) 28(r) 24(w) 2(r) 20(r) 4(r) 18(w) 3(w) 8(w) 25(w) 19(w) 26(r) 20(r) 13(w) 17(r) 28(w) 1(w) 27(r) 16(w) 3(r) 28(r) 14(w) 21(w) 12(r) 14(w) 16(w) 2(r) 30(r) 29(w) 11(r) 30(r)
  22(r) 8(r) 15(r) 23(r) 27(w) 10(r) 2(w) 30(r) 18(w) 27(r) 10(w) 5(w) 8(w) 14(r) 21(r) 6(r) 14(r) 10(r) 13(r) 8(r) 7(w) 18(w) 28(w) 10(w) 1(r) 13(r) 3(w) 30(r) 4(r) 13(w) 21(w) 25(r) 21(r) 6(w) 9(r)
  9(w) 15(w) 10(w) 30(w) 24(w) 28(w) 9(r) 29(r) 6(r) 14(w) 11(r) 3(w) 27(w) 20(w) 7(w) 26(w) 18(r) 24(r) 23(r) 20(w) 24(r) 28(r) 22(r) 23(r) 1(r) 5(w) 5(w) 17(r) 25(w) 18(r) 17(r) 25(r) 17(r) 18(w) 25(w)
  10(w) 16(w) 25(r) 8(w) 13(w) 1(w) 11(r) 7(r) 19(r) 30(w) 13(w) 15(w) 10(w) 6(w) 7(w) 29(r) 21(w) 26(w) 20(r) 5(r) 18(r) 16(w) 9(w) 26(r) 2(w) 19(r) 12(w) 27(w) 27(r) 22(r) 13(w) 6(w) 7(w) 29(w) 6(r)
  11(w) 29(r) 16(r) 9(w) 18(r) 7(w) 21(w) 2(w) 8(r) 18(r) 30(r) 6(r) 8(r) 26(w) 18(r) 12(r) 5(w) 3(r) 20(w) 1(r) 5(w) 30(w) 12(w) 23(r) 26(w) 3(w) 5(w) 2(r) 1(w) 3(r) 29(w) 3(r) 2(r) 6(w) 11(w) 11(r)
  12(w) 23(r) 4(w) 12(w) 10(w) 3(w) 17(r) 9(w) 20(w)
```

(4) 결과

- (1),(2),(3)에서 입력한 내용을 바탕으로 페이지 교체 알고리즘을 시뮬레이션하여 출력한다.
- 결과는 알고리즘 이름, 페이지 교체 내용 및 페이지 폴트 여부, 페이지폴트 수 순으로 출력된다.

```
Optimal
11:     |11     |       |       |F
19:     |11     |19     |       |F
26:     |11     |19     |26     |F
27:     |11     |19     |27     |F
13:     |11     |19     |13     |F
23:     |11     |19     |23     |F
22:     |11     |19     |22     |F
25:     |11     |19     |25     |F
6:      |6      |19     |25     |F
25:     |6      |19     |25
9:      |9      |19     |25     |F
```

(중략)

```
16:     |22     |17     |16     |F
2:      |22     |17     |2      |F
8:      |22     |17     |8      |F
17:     |22     |17     |8
22:     |22     |17     |8
3:      |22     |3      |8      |F
OPT Page Fault: 371
```

- 여러개의 알고리즘을 선택했거나 8번을 선택한 경우, 해당 알고리즘들이 차례대로 위의 결과를 출력한다.

- 출력된 시뮬레이션 결과는 save.txt 파일에 저장된다.

```
  1 Optimal
  2 11:     |11     |       |       |F
  3 19:     |11     |19     |       |F
  4 26:     |11     |19     |26     |F
  5 27:     |11     |19     |27     |F
  6 13:     |11     |19     |13     |F
  7 23:     |11     |19     |23     |F
  8 22:     |11     |19     |22     |F
  9 25:     |11     |19     |25     |F
 10 6:      |6      |19     |25     |F
 11 25:     |6      |19     |25
 12 9:      |9      |19     |25     |F
 13 12:     |12     |19     |25     |F
 14 30:     |30     |19     |25     |F
 15 19:     |30     |19     |25
 16 3:      |30     |3      |25     |F
 17 11:     |30     |11     |25     |F
 18 17:     |30     |17     |25     |F
 19 16:     |30     |16     |25     |F
 20 10:     |10     |16     |25     |F
 21 20:     |20     |16     |25     |F
 22 15:     |15     |16     |25     |F
 23 28:     |28     |16     |25     |F
 24 4:      |4      |16     |25     |F
 25 25:     |4      |16     |25
 26 14:     |4      |16     |14     |F
 27 14:     |4      |16     |14
 28 24:     |4      |16     |24     |F
 29 27:     |4      |27     |24     |F
```

(중략)

```
493 26:     |22     |17     |26     |F
494 17:     |22     |17     |26
495 22:     |22     |17     |26
496 16:     |22     |17     |16     |F
497 2:      |22     |17     |2      |F
498 8:      |22     |17     |8      |F
499 17:     |22     |17     |8
500 22:     |22     |17     |8
501 3:      |22     |3      |8      |F
502 OPT Page Fault: 371
503
```

- 결과의 출력 및 저장이 완료되면 프로그램을 종료한다.

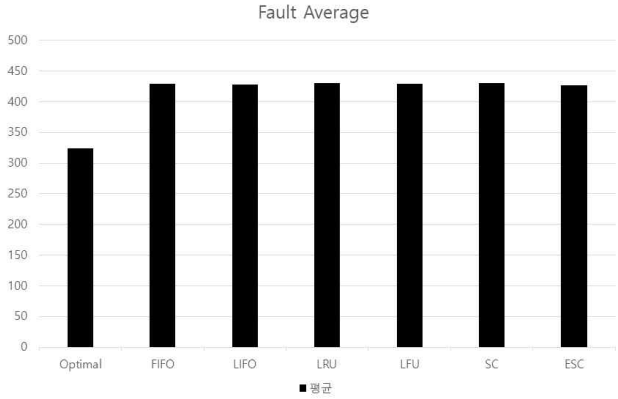
```
22:     |11     |19     |22     |F
3:      |11     |19     |3      |F
LIFO Page Fault: 457

D. 종료.
```

5. 알고리즘 성능 분석 비교

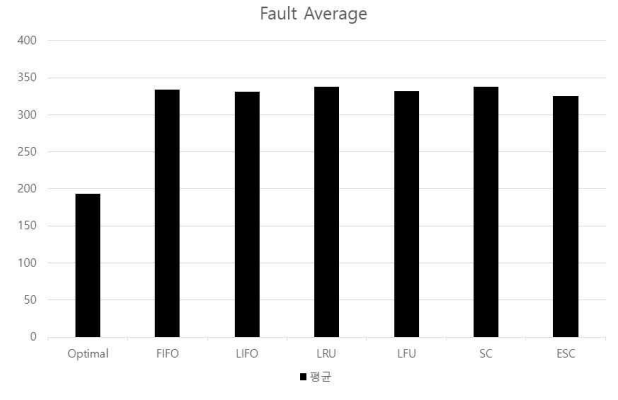
(1) 프레임수 4, 스트링길이 500 10회 실험 Fault 수 비교

	1	2	3	4	5	6	7	8	9	10	평균
Optimal	325	330	328	324	328	311	324	321	328	322	324.10
FIFO	432	442	432	431	428	422	424	426	435	427	429.90
LIFO	438	427	416	416	429	442	423	427	432	432	428.20
LRU	428	444	435	432	429	423	426	423	442	427	430.90
LFU	437	429	422	421	435	426	429	434	426	435	429.40
SC	431	444	433	432	429	422	425	425	437	428	430.60
ESC	427	440	429	428	426	417	421	418	438	424	426.80



(2) 프레임수 10, 스트링길이 500 10회 실험 Fault 수 비교

	1	2	3	4	5	6	7	8	9	10	평균
Optimal	191	202	192	190	196	188	187	203	200	184	193.30
FIFO	332	348	330	324	346	330	322	352	341	317	334.20
LIFO	326	325	345	334	324	336	339	332	315	330	330.60
LRU	334	351	340	322	357	325	326	353	345	327	338.00
LFU	341	337	332	321	343	317	329	339	341	322	332.20
SC	334	350	335	325	355	328	324	355	344	325	337.50
ESC	326	331	327	316	342	313	320	335	330	315	325.50



(3) 결론

프레임 수 4, 페이지스트링 길이 500으로 알고리즘을 실험했을 때, OPT를 제외하고는 거의 비슷한 결과를 보인다. 프레임 수를 늘려, 프레임 수 10, 페이지스트링 길이 500으로 알고리즘을 실험했을 때 또한, OPT를 제외하고는 거의 비슷한 결과를 보인다.

OPT를 제외한 알고리즘 중에서는 평균적으로 ESC가 가장 적은 Fault 수를 보인다. ESC를 제외한 다른 알고리즘은 상대적으로 큰 차이를 보이지는 못했다. 작은 수치이지만 두 조건 모두에서 LFU가 많은 Fault 수를 보이고, LIFO가 적은 Fault 수를 보였다.

실험 상에서 역시 Optimal이 압도적인 성능을 보였다. 나머지 알고리즘 중에서 ESC가 상대적으로 좋은 성능을 보였으며, 이를 제외한 다른 알고리즘은 비슷한 성능을 보였다.