1 .Write the simulation program for demand paging and show the page scheduling and total number of page faults according the MRU page replacement algorithm. Assume the memory ofn frames. ReferenceString:8,5,7,8,5,7,2,3,7,3,5,9,4,6,2

```
#include <stdio.h>
int findMRU(int recent[], int n) {
    int max = recent[0], pos = 0;
    for (int i = 1; i < n; i++) {
        if (recent[i] > max) {
            max = recent[i];
            pos = i;
        }
    }
    return pos;
}

int main() {
    int frames[10], recent[10], refStr[] = {8,5,7,8,5,7,2,3,7,3,5,9,4,6,2};
    int n = 15;
    int no_of_frames, page_faults = 0, time = 0;
    printf("Enter number of frames: ");
    scanf("%d", &no_of_frames);
    for (int i = 0; i < no_of_frames; i++) {
        frames[i] = -1;
        recent[i] = 0;
```

```
}
printf("\nReference String: ");
for (int i = 0; i < n; i++)
   printf("%d ", refStr[i]);
printf("\n\nPage Replacement Process (MRU):\n");
for (int i = 0; i < n; i++) {
   int page = refStr[i];
   int found = 0;
   for (int j = 0; j < no_of_frames; j++) {
      if (frames[j] == page) {
         found = 1;
         recent[j] = ++time;
         break;
      }
   }
   if (!found) {
      int empty = -1;
      for (int j = 0; j < no_of_frames; j++) {
         if (frames[j] == -1) {
            empty = j;
            break;
         }
      }
      if (empty != -1) {
```

```
            frames[empty] = page;
            recent[empty] = ++time;
        }
        else {
            int pos = findMRU(recent, no_of_frames);
            frames[pos] = page;
            recent[pos] = ++time;
        }

        page_faults++;
    }
    printf("After reference %2d: ", page);
    for (int j = 0; j < no_of_frames; j++) {
        if (frames[j] != -1)
            printf("%d ", frames[j]);
        else
            printf("- ");
    }
    if (!found)
        printf(" (Page Fault)");
    printf("\n");
}
printf("\nTotal Page Faults = %d\n", page_faults);
return 0;
```

}

2 .Write the simulation program for demand paging and show the page scheduling and total number of page faults according the FIFO page replacement algorithm. Assume the memory of n frames. ReferenceString:3,4,5,6,3,4,7,3,4,5,6,7,2,4,6.

```
#include <stdio.h>

int main() {
   int frames[10], refStr[] = {3,4,5,6,3,4,7,3,4,5,6,7,2,4,6};
   int n = 15;
   int no_of_frames;
   int page_faults = 0;
   int front = 0;


   printf("Enter number of frames: ");
   scanf("%d", &no_of_frames);
   for (int i = 0; i < no_of_frames; i++)
      frames[i] = -1;
   printf("\nReference String: ");
   for (int i = 0; i < n; i++)
      printf("%d ", refStr[i]);
   printf("\n\nPage Replacement Process (FIFO):\n");
   for (int i = 0; i < n; i++) {
      int page = refStr[i];
      int found = 0;
```

```
    for (int j = 0; j < no_of_frames; j++) {
        if (frames[j] == page) {
            found = 1;
            break;
        }
    }
    if (!found) {
        frames[front] = page;
        front = (front + 1) % no_of_frames;  // move to next frame (FIFO)
        page_faults++;
    }
    printf("After reference %2d: ", page);
    for (int j = 0; j < no_of_frames; j++) {
        if (frames[j] != -1)
            printf("%d ", frames[j]);
        else
            printf("- ");
    }
    if (!found)
        printf(" (Page Fault)");
    printf("\n");
}
printf("\nTotal Page Faults = %d\n", page_faults);
return 0;
```

```
}
```

3 .Write the simulation program to implement demand paging and show the page scheduling and total number of page faults according to the LRU (using counter method) page replacement algorithm. Assume the memory of n frames.
ReferenceString: 3,5,7,2,5, 1,2,3, 1,3,5,3, 1,6,2.

```
#include <stdio.h>
int findLRU(int counter[], int n) {
    int min = counter[0], pos = 0;
    for (int i = 1; i < n; i++) {
        if (counter[i] < min) {
            min = counter[i];
            pos = i;
        }
    }
    return pos;
}
int main() {
    int frames[10], counter[10];
    int refStr[] = {3,5,7,2,5,1,2,3,1,3,5,3,1,6,2};
    int n = 15;  // Length of reference string
    int no_of_frames, page_faults = 0, time = 0;
    printf("Enter number of frames: ");
    scanf("%d", &no_of_frames);
```

```
for (int i = 0; i < no_of_frames; i++) {
    frames[i] = -1;
    counter[i] = 0;
}
printf("\nReference String: ");
for (int i = 0; i < n; i++)
    printf("%d ", refStr[i]);
printf("\n\nPage Replacement Process (LRU using Counter Method):\n");
for (int i = 0; i < n; i++) {
    int page = refStr[i];
    int found = 0;

    for (int j = 0; j < no_of_frames; j++) {
        if (frames[j] == page) {
            found = 1;
            counter[j] = ++time;  // Update last used time
            break;
        }
    }
    if (!found) {
        int empty = -1;
        for (int j = 0; j < no_of_frames; j++) {
            if (frames[j] == -1) {
                empty = j;
```

```
            break;
        }
    }
    if (empty != -1) {
        frames[empty] = page;
        counter[empty] = ++time;
    } else {
        int pos = findLRU(counter, no_of_frames);
        frames[pos] = page;
        counter[pos] = ++time;
    }

    page_faults++;
}
printf("After reference %2d: ", page);
for (int j = 0; j < no_of_frames; j++) {
    if (frames[j] != -1)
        printf("%d ", frames[j]);
    else
        printf("- ");
}
if (!found)
    printf(" (Page Fault)");
printf("\n");
```

```
    }
    printf("\nTotal Page Faults = %d\n", page_faults);
    return 0;
}
```

4 .Write the simulation program for demand paging and show the page scheduling and total number of page faults according the MFU page replacement algorithm. Assume the memory of n frames. ReferenceString:8,5,7,8,5,7,2,3, 7,3,5,9,4,6,2.

```
#include <stdio.h>
int findMFU(int freq[], int n) {
   int max = freq[0], pos = 0;
   for (int i = 1; i < n; i++) {
      if (freq[i] > max) {
         max = freq[i];
         pos = i;
      }
   }
   return pos;
}

int main() {
   int frames[10], freq[10];
   int refStr[] = {8,5,7,8,5,7,2,3,7,3,5,9,4,6,2};
   int n = 15;  // length of reference string
```

```
int no_of_frames, page_faults = 0;
printf("Enter number of frames: ");
scanf("%d", &no_of_frames);
for (int i = 0; i < no_of_frames; i++) {
   frames[i] = -1;
   freq[i] = 0;
}

printf("\nReference String: ");
for (int i = 0; i < n; i++)
   printf("%d ", refStr[i]);
printf("\n\nPage Replacement Process (MFU):\n");
for (int i = 0; i < n; i++) {
   int page = refStr[i];
   int found = 0;
   for (int j = 0; j < no_of_frames; j++) {
      if (frames[j] == page) {
         found = 1;
         freq[j]++;
         break;
      }
   }
   if (!found) {
      int empty = -1;
```

```
    for (int j = 0; j < no_of_frames; j++) {
        if (frames[j] == -1) {
            empty = j;
            break;
        }
    }
    if (empty != -1) {
        frames[empty] = page;
        freq[empty] = 1;  // initialize frequency
    } else {
        int pos = findMFU(freq, no_of_frames);
        frames[pos] = page;
        freq[pos] = 1;
    }
    page_faults++;
}
printf("After reference %2d: ", page);
for (int j = 0; j < no_of_frames; j++) {
    if (frames[j] != -1)
        printf("%d ", frames[j]);
    else
        printf("- ");
}
if (!found)
```

```
            printf(" (Page Fault)");
        printf("\n");
    }

    printf("\nTotal Page Faults = %d\n", page_faults);
    return 0;
}
```

5 . Write a C program to implement the shell which displays the command prompt "myshell$". It accepts the command, tokenize the command line and execute it by creating the child process. Also implement the additional command
'typeline' as

typeline+nfilename:-To print first n lines in the file.
typeline -afilename:-To print all lines in the file.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

#define MAX 100

void typeline(char *option, char *filename) {
```

```
    FILE *fp;
    char line[256];
    int count = 0;

    fp = fopen(filename, "r");
    if (fp == NULL) {
        printf("File not found: %s\n", filename);
        return;
    }

    // typeline +n filename → print first n lines
    if (option[0] == '+') {
        int n = atoi(option + 1);
        while (fgets(line, sizeof(line), fp) != NULL && count < n) {
            printf("%s", line);
            count++;
        }
    }

    else if (strcmp(option, "-a") == 0) {
        while (fgets(line, sizeof(line), fp) != NULL) {
            printf("%s", line);
        }
    } else {
        printf("Invalid typeline command format.\n");
        printf("Usage:\n  typeline +n filename  → Print first n lines\n");
        printf("  typeline -a filename  → Print all lines\n");
    }

    fclose(fp);
}

int main() {
    char input[MAX], *args[10];
    char *token;
    int i;

    while (1) {
        printf("myshell$ ");
        fflush(stdout);
```

```
if (fgets(input, sizeof(input), stdin) == NULL)
    break;
input[strcspn(input, "\n")] = '\0'; // remove newline

if (strlen(input) == 0)
    continue;

if (strcmp(input, "exit") == 0)
    break;

i = 0;
token = strtok(input, " ");
while (token != NULL) {
    args[i++] = token;
    token = strtok(NULL, " ");
}
args[i] = NULL;
if (strcmp(args[0], "typeline") == 0) {
    if (i == 3)
        typeline(args[1], args[2]);
    else
        printf("Usage:\n  typeline +n filename\n  typeline -a filename\n");
}

else {
    pid_t pid = fork();

    if (pid == 0) {
        // Child process executes the command
        if (execvp(args[0], args) == -1) {
            perror("Command execution failed");
        }
        exit(0);
    }
    else if (pid > 0) {

        wait(NULL);
    }
    else {
        perror("Fork failed");
```

```
        }
      }
   }

   printf("Exiting myshell...\n");
   return 0;
}
```

6 .Write a progranto implement the toy shell. It should display the command prompt "myshell$". Tokenize the command line and execute the given command by creating the child process.Additionally its hould interpret the following commands.

count c filename-To print number of characters in the file.

count wfilename:- To print number of words in the file.

countl filename :-To print number of lines in the file.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

#define MAX 100

void countCommand(char *option, char *filename) {
   FILE *fp;
   char ch;
   int lines = 0, words = 0, chars = 0;
   int inWord = 0;

   fp = fopen(filename, "r");
   if (fp == NULL) {
      printf("File not found: %s\n", filename);
      return;
```

```
    }

    while ((ch = fgetc(fp)) != EOF) {
        chars++;

        if (ch == ' ' || ch == '\n' || ch == '\t')
            inWord = 0;
        else if (inWord == 0) {
            inWord = 1;
            words++;
        }

        if (ch == '\n')
            lines++;
    }

    fclose(fp);

    if (strcmp(option, "c") == 0)
        printf("Number of characters: %d\n", chars);
    else if (strcmp(option, "w") == 0)
        printf("Number of words: %d\n", words);
    else if (strcmp(option, "l") == 0)
        printf("Number of lines: %d\n", lines);
    else
        printf("Invalid count option. Use c, w, or l.\n");
}
int main() {
    char input[MAX], *args[10];
    char *token;
    int i;

    while (1) {
        printf("myshell$ ");
        fflush(stdout);
```

```
        if (fgets(input, sizeof(input), stdin) == NULL)
            break;
        input[strcspn(input, "\n")] = '\0'; // remove newline

        if (strlen(input) == 0)
            continue;


        if (strcmp(input, "exit") == 0)
            break;


        i = 0;
        token = strtok(input, " ");
        while (token != NULL) {
            args[i++] = token;
            token = strtok(NULL, " ");
        }
        args[i] = NULL;


        if (strcmp(args[0], "count") == 0) {
            if (i == 3)
                countCommand(args[1], args[2]);
            else
                printf("Usage:\n  count c filename\n  count w filename\n
count l filename\n");
        }

        else {
            pid_t pid = fork();

            if (pid == 0) {
                // Child executes command
```

```
            if (execvp(args[0], args) == -1)
                perror("Command execution failed");
            exit(0);
        }
        else if (pid > 0) {

            wait(NULL);
        }
        else {
            perror("Fork failed");
        }
    }
}

    printf("Exiting myshell...\n");
    return 0;      }
```

7 .Write a program to implement the shell. It should display the command prompt"myshell$". Tokenize the command line and execute the given command bycreating the child process. Additionally it should interpret the following commands.

myshellS search c file name pattern filename pattern:-To display first occurrence of pattern in the file.

myshellS search c file name pattern :-To count the number of occurrence of pattern in the file.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

#define MAX 100
```

```
void searchCommand(char *option, char *pattern, char *filename) {
    FILE *fp;
    char line[256];
    int count = 0, found = 0, lineNumber = 0;

    fp = fopen(filename, "r");
    if (fp == NULL) {
        printf("File not found: %s\n", filename);
        return;
    }

    while (fgets(line, sizeof(line), fp) != NULL) {
        lineNumber++;
        char *pos = strstr(line, pattern);
        if (pos != NULL) {
            count++;
            if (strcmp(option, "f") == 0) {  // display first occurrence
                printf("First occurrence of '%s' found in line %d:\n", pattern,
lineNumber);
                printf("%s", line);
                fclose(fp);
                return;
            }

            char *temp = pos + 1;
            while ((temp = strstr(temp, pattern)) != NULL) {
                count++;
                temp++;
            }
        }
    }

    if (strcmp(option, "c") == 0)
        printf("Total occurrences of '%s': %d\n", pattern, count);
    else if (strcmp(option, "f") != 0)
```

```
        printf("Invalid option. Use 'f' for first occurrence or 'c' for count.\n");

    fclose(fp);
}
int main() {
    char input[MAX], *args[10];
    char *token;
    int i;

    while (1) {
        printf("myshell$ ");
        fflush(stdout);
        if (fgets(input, sizeof(input), stdin) == NULL)
            break;
        input[strcspn(input, "\n")] = '\0'; // remove newline

        if (strlen(input) == 0)
            continue;

        if (strcmp(input, "exit") == 0)
            break;

        i = 0;
        token = strtok(input, " ");
        while (token != NULL) {
            args[i++] = token;
            token = strtok(NULL, " ");
        }
        args[i] = NULL;
        if (strcmp(args[0], "search") == 0) {
            if (i == 4)
                searchCommand(args[1], args[2], args[3]);
            else
                printf("Usage:\n  search f pattern filename   → first occurrence\n
search c pattern filename   → count occurrences\n");
        }
```

```
        else {
            pid_t pid = fork();

            if (pid == 0) {

                if (execvp(args[0], args) == -1)
                    perror("Command execution failed");
                exit(0);
            }
            else if (pid > 0) {
                // Parent waits for child
                wait(NULL);
            }
            else {
                perror("Fork failed");
            }
        }
    }

    printf("Exiting myshell...\n");
    return 0;
}
```