Demand Paging Simulation using FIFO Page Replacement Algorithm:

```
#include <stdio.h>
#define MAX 20

int frames[MAX], ref[MAX], mem[MAX][MAX], faults = 0, sp = 0, m, n;

void accept() {
   printf("Enter number of frames: ");
   scanf("%d", &n);
   printf("Enter number of references: ");
   scanf("%d", &m);
   printf("Enter reference string:\n");
   for (int i = 0; i < m; i++) {
      printf("[%d] = ", i);
      scanf("%d", &ref[i]);
   }
}

int search(int pno) {
   for (int i = 0; i < n; i++) {
      if (frames[i] == pno) return i;  // If page found in frame, return index
   }
   return -1;  // Page not found
}

void fifo() {
   for (int i = 0; i < m; i++) {
      if (search(ref[i]) == -1) {  // If page not found in frame
         frames[sp] = ref[i];     // Replace page at the current pointer (FIFO order)
         sp = (sp + 1) % n;       // Move pointer in a circular fashion
         faults++;                // Increment page faults
      }
```

```
        for (int j = 0; j < n; j++) {
            mem[j][i] = frames[j];   // Store current memory state
        }
    }
}

void disp() {
    printf("\nReference String:\n");
    for (int i = 0; i < m; i++) {
        printf("%3d", ref[i]);  // Display reference string
    }

    printf("\n\nFrame Allocation:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            if (mem[i][j]) {
                printf("%3d", mem[i][j]);  // Display memory frames at each step
            } else {
                printf("   ");
            }
        }
        printf("\n");
    }

    printf("\nTotal Page Faults: %d\n", faults);  // Display total page faults
}

int main() {
    accept();  // Accept inputs
    fifo();    // Perform FIFO page replacement
    disp();    // Display the results
    return 0;
```

```
}
```

Program 2

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

void search_file(char *option, char *filename, char *pattern) {
    FILE *file = fopen(filename, "r");
    if (file == NULL) {
        printf("File %s not found.\n", filename);
        return;
    }

    char line[256];
    int count = 0, line_num = 0;

    while (fgets(line, sizeof(line), file)) {
        line_num++;
        if (strstr(line, pattern) != NULL) {
            if (strcmp(option, "a") == 0) {
                printf("Line %d: %s", line_num, line);
            }
            count++;
        }
    }

    if (strcmp(option, "c") == 0) {
        printf("Pattern '%s' occurred %d times in file %s.\n", pattern, count, filename);
    }

    fclose(file);
}
```

```
int main() {
    char command[100], *args[10];

    while (1) {
        printf("\nmyshell$ ");
        fgets(command, 100, stdin);
        command[strlen(command) - 1] = '\0';  // Remove newline character

        char *token = strtok(command, " ");
        int i = 0;
        while (token != NULL) {
            args[i++] = token;
            token = strtok(NULL, " ");
        }
        args[i] = NULL;

        if (strcmp(args[0], "search") == 0) {
            search_file(args[1], args[2], args[3]);
        } else if (strcmp(args[0], "exit") == 0) {
            exit(0);
        } else {
            int pid = fork();
            if (pid == 0) {  // Child process
                execvp(args[0], args);
                exit(0);
            } else {  // Parent process
                wait(NULL);
            }
        }
    }
```

```
    return 0;
}
```