Dining philosopher problem

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>  // For abs()

int tph, philname[20], status[20], howhung, hu[20], cho; // Global variables

void one();  // Function declaration
void two();  // Function declaration

int main() {
    int i;
    printf("\n\nDINING PHILOSOPHER PROBLEM");
    printf("\nEnter the total no. of philosophers: ");
    scanf("%d", &tph);

    for (i = 0; i < tph; i++) {
        philname[i] = (i + 1); // Assigning philosopher numbers
        status[i] = 1; // Setting all philosophers' status as thinking (1)
    }

    printf("How many are hungry: ");
    scanf("%d", &howhung);

    if (howhung == tph) {
        printf("\nAll are hungry.. Deadlock stage will occur\n");
        printf("Exiting..\n");
        exit(0); // Exiting due to deadlock
    } else {
        for (i = 0; i < howhung; i++) {
            printf("Enter philosopher %d position: ", (i + 1));
            scanf("%d", &hu[i]);
            status[hu[i]] = 2; // Set status as hungry (2)
        }

        do {
            printf("\n1. One can eat at a time\t2. Two can eat at a time\t3. Exit\nEnter your choice: ");
            scanf("%d", &cho);

            switch (cho) {
                case 1:
                    one();
                    break;
                case 2:
                    two();
                    break;
                case 3:
                    exit(0);
                default:
                    printf("\nInvalid option..");
            }
        } while (1);
    }

    return 0;
}

void one() {
    int pos = 0, i, x;
    printf("\nAllow one philosopher to eat at any time\n");

    for (i = 0; i < howhung; i++, pos++) {
        printf("\nP %d is granted to eat", philname[hu[pos]]);
        status[hu[pos]] = 1; // After eating, status becomes thinking (1)

        for (x = pos + 1; x < howhung; x++) {
            printf("\nP %d is waiting", philname[hu[x]]);
        }

        printf("\nP %d finished eating", philname[hu[pos]]);
    }
}

void two() {
    int i, j, s = 0, t, r, x;
    printf("\nAllow two philosophers to eat at the same time\n");

    for (i = 0; i < howhung; i++) {
        for (j = i + 1; j < howhung; j++) {
            if (abs(hu[i] - hu[j]) >= 1 && abs(hu[i] - hu[j]) != tph - 1) {
                printf("\n\nCombination %d\n", (s + 1));
                t = hu[i];
                r = hu[j];
                s++;
                printf("\nP %d and P %d are granted to eat", philname[t], philname[r]);

                for (x = 0; x < howhung; x++) {
                    if (hu[x] != t && hu[x] != r) {
                        printf("\nP %d is waiting", philname[hu[x]]);
                    }
                }

                printf("\nP %d and P %d finished eating", philname[t], philname[r]);
            }
        }
    }
}
```

Producer Consumer

```
#include<stdio.h>

void main() {
    int buffer[10], bufsize, in, out, produce, consume, choice = 0;
    in = 0;
    out = 0;
    bufsize = 10;

    while(choice != 3) {
        printf("\n1. Produce \t 2. Consume \t3. Exit");
        printf("\nEnter your choice: ");
        scanf("%d", &choice);

        switch(choice) {
            case 1:
                if((in+1) % bufsize == out) {
                    printf("\nBuffer is Full");
                } else {
                    printf("\nEnter the value: ");
                    scanf("%d", &produce);
                    buffer[in] = produce;
                    in = (in+1) % bufsize;
                }
                break;

            case 2:
                if(in == out) {
                    printf("\nBuffer is Empty");
                } else {
                    consume = buffer[out];
                    printf("\nThe consumed value is %d", consume);
                    out = (out+1) % bufsize;
                }
                break;

            case 3:
                printf("Exiting...");
                break;

            default:
                printf("\nInvalid choice");
        }
    }
}
```

Fifo page replacement

```
#include<stdio.h>

#include<conio.h>


int main()

{

    int i, j, k, f, pf = 0, count = 0, rs[25], m[10], n;


    printf("\nEnter the length of reference string -- ");

    scanf("%d", &n);


    printf("\nEnter the reference string -- ");

    for(i = 0; i < n; i++)

        scanf("%d", &rs[i]);


    printf("\nEnter number of frames -- ");

    scanf("%d", &f);


    // Initialize all frames to -1 (empty)

    for(i = 0; i < f; i++)

        m[i] = -1;


    printf("\nThe Page Replacement Process is -- \n");

    for(i = 0; i < n; i++)

    {

        // Check if the page is already in any of the frames

        for(k = 0; k < f; k++)
        {
            if(m[k] == rs[i])
                break;
        }

        // If page not found, page fault occurs
        if(k == f)
        {
            // Replace the page in FIFO manner
            m[count] = rs[i];
            count = (count + 1) % f; // To cycle through frames
            pf++; // Increment page fault count

            // Display the frame contents
            for(j = 0; j < f; j++)
                printf("\t%d", m[j]);

            printf("\tPF No. %d", pf);
        }
        else
        {
            // If no page fault, display the frame contents without page fault increment
            for(j = 0; j < f; j++)
                printf("\t%d", m[j]);
        }
        printf("\n");
    }

    printf("\nThe number of Page Faults using FIFO are %d", pf);
    getch();

    return 0;
}
```

Optimal page replacement

```
#include<stdio.h>

int n; // Number of frames

// Function to find the position of the maximum value in an array
int findmax(int a[]) {
    int max, i, k = 0;
    max = a[0];
    for(i = 1; i < n; i++) { // i should start from 1, not 0
        if(max < a[i]) {
            max = a[i];
            k = i;
        }
    }
    return k;
}

int main() {
    int seq[30], fr[5], pos[5], find, flag, max, i, j, m, k, t, s;
    int count = 1, pf = 0, p = 0;
    float pfr;
```

```
printf("Enter maximum limit of the sequence: ");
scanf("%d", &max);

printf("\nEnter the sequence: ");
for(i = 0; i < max; i++)
    scanf("%d", &seq[i]);

printf("\nEnter number of frames: ");
scanf("%d", &n);

// Initialize the first frame with the first page from the sequence
fr[0] = seq[0];
pf++; // First page fault
printf("%d\t", fr[0]);

i = 1;
while(count < n) {
    flag = 1;
    p++;

    // Check if the page is already in a frame
    for(j = 0; j < i; j++) {
        if(seq[i] == seq[j]) {
            flag = 0; // Page is already present
            break;
        }
    }
    // If page is not found, insert it into the next available frame
    if(flag != 0) {
        fr[count] = seq[i];
        printf("%d\t", fr[count]);
        count++;
        pf++; // Page fault occurred
    }
    i++;
}

printf("\n");

// Continue the sequence after initializing the frames
for(i = p; i < max; i++) {
    flag = 1;

    // Check if the page is already in one of the frames
    for(j = 0; j < n; j++) {
        if(seq[i] == fr[j]) {
            flag = 0; // Page is already present
            break;
        }
    }

    if(flag != 0) {
        // Find the page to replace using the optimal algorithm
        for(j = 0; j < n; j++) {
            m = fr[j];
            pos[j] = -1; // Initialize with a value that means the page is not found in the future
    for(k = i + 1; k < max; k++) {
        if(seq[k] == m) {
            pos[j] = k; // Store the future position of the page
            break;
        }
    }
}

// Check if there's any page that won't be needed in the future
flag = 1;
for(k = 0; k < n; k++) {
    if(pos[k] == -1) {
        s = k;
        flag = 0;
        break;
    }
}

if(flag != 0) {
    // If all pages are used in the future, replace the one used the farthest in the future
    s = findmax(pos);
}

fr[s] = seq[i]; // Replace the selected page
for(k = 0; k < n; k++)
    printf("%d\t", fr[k]);
printf("\n");
pf++; // Page fault occurred
```

```
        }
    }

    pfr = (float)pf / (float)max;
    printf("\nThe number of page faults is %d", pf);
    printf("\nPage fault rate: %f\n", pfr);

    return 0;
}
```

Lru page replacement

```
#include<stdio.h>
#define high 37

void main()
{
    int fframe[10], used[10], index = 0;
    int count, n1, k, nf, np = 0, page[high], tmp;
    int flag = 0, pf = 0;

    // Taking number of frames input
    printf("Enter the number of frames: ");
    scanf("%d", &nf);

    // Initializing frame array
    for(count = 0; count < nf; count++)
        fframe[count] = -1;
```

```
// Taking page inputs
printf("LRU page replacement algorithm in C\n");
printf("Enter pages (press -999 to exit):\n");
for(count = 0; count < high; count++)
{
    scanf("%d", &tmp);
    if(tmp == -999) break;
    page[count] = tmp;
    np++;
}

// Implementing LRU page replacement algorithm
for(count = 0; count < np; count++)
{
    flag = 0;

    // Check if the page is already in a frame
    for(n1 = 0; n1 < nf; n1++)
    {
        if(fframe[n1] == page[count])
        {
            flag = 1;
            break;
        }
    }

    // Page fault occurs if the page is not in any frame
    if(flag == 0)
```

```
{
    for(n1 = 0; n1 < nf; n1++)
        used[n1] = 0;

    // Find the least recently used frame
    for(n1 = 0, tmp = count - 1; n1 < nf - 1 && tmp >= 0; n1++, tmp--)
    {
        for(k = 0; k < nf; k++)
        {
            if(fframe[k] == page[tmp])
                used[k] = 1;
        }
    }

    for(n1 = 0; n1 < nf; n1++)
    {
        if(used[n1] == 0)
        {
            index = n1;
            break;
        }
    }

    fframe[index] = page[count];
    printf("\nFault: ");
    pf++;
}

// Print current frame status
```

```
        for(k = 0; k < nf; k++)
            printf("%d\t", fframe[k]);
    }

    printf("\nTotal number of page faults: %d\n", pf);
}
```