FIFO
Method1:

```
#include <stdio.h>

#define FRAME_SIZE 3

int main() {
   int page_reference_string[] = {1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5}; // example page reference
string
   int page_faults = 0; // count of page faults
   int frames[FRAME_SIZE]; // frames array
   int oldest_page_index = 0; // index of the oldest page in frames
   int i, j, k;

   // initialize frames array with -1 (representing empty frame)
   for (i = 0; i < FRAME_SIZE; i++) {
      frames[i] = -1;
   }

   // iterate through page reference string
   for (i = 0; i < sizeof(page_reference_string) / sizeof(page_reference_string[0]); i++) {
      int page = page_reference_string[i];
      int page_found = 0;

      // check if page is already in frames array
      for (j = 0; j < FRAME_SIZE; j++) {
         if (frames[j] == page) {
            page_found = 1;
            break;
         }
      }

      // if page is not in frames array, replace oldest page with new page
      if (!page_found) {
         frames[oldest_page_index] = page;
         oldest_page_index = (oldest_page_index + 1) % FRAME_SIZE;
         page_faults++;

         // print current state of frames array
         for (k = 0; k < FRAME_SIZE; k++) {
            printf("%d ", frames[k]);
         }
         printf("\n");
      }
```

```
    }

    // print total number of page faults
    printf("Total page faults: %d\n", page_faults);

    return 0;
}
```

1. First fit; worst fit; best fit:

Method-1:

```
#include <stdio.h>
#include <stdlib.h>
#define max 25
int main()
{
   int choice;
   int frag[max], b[max], f[max], i, j, nb, nf, temp, highest = 0,lowest=10000;
   static int bf[max], ff[max];
   do
   {
      printf("\n\n");
      printf("Memory Allocation Techniques:\n");
      printf("1. Worst-fit\n");
      printf("2. Best-fit\n");
      printf("3. First-fit\n");
      printf("4. Exit\n");
      printf("Enter your choice: ");
      scanf("%d", &choice);

      switch (choice)
      {
      case 1:
         printf("\n\tMemory Management Scheme - Worst Fit");
         printf("\nEnter the number of blocks:");
         scanf("%d", &nb);
         printf("Enter the number of files:");
         scanf("%d", &nf);
         printf("\nEnter the size of the blocks:-\n");
         for (i = 1; i <= nb; i++)
         {
            printf("Block %d:", i);
            scanf("%d", &b[i]);
         }
```

```
    printf("Enter the size of the files :-\n");
    for (i = 1; i <= nf; i++)
    {
       printf("File %d:", i);
       scanf("%d", &f[i]);
    }
    for (i = 1; i <= nf; i++)
    {
       for (j = 1; j <= nb; j++)
       {
          if (bf[j] != 1) // if bf[j] is not allocated
          {
             temp = b[j] - f[i];
             if (temp >= 0)
                if (highest < temp)
                {
                   ff[i] = j;
                   highest = temp;
                }
          }
       }
       frag[i] = highest;
       bf[ff[i]] = 1;
       highest = 0;
    }
    printf("\nFile_no:\tFile_size :\tBlock_no:\tBlock_size:\tFragement");
    for (i = 1; i <= nf; i++)
       printf("\n%d\t\t%d\t\t\t%d\t\t\t%d\t\t\t%d", i, f[i], ff[i], b[ff[i]], frag[i]);
    break;
 case 2:
    printf("\nEnter the number of blocks:");
    scanf("%d", &nb);
    printf("Enter the number of files:");
    scanf("%d", &nf);
    printf("\nEnter the size of the blocks:-\n");
    for (i = 1; i <= nb; i++)
    {
       printf("Block %d:", i);
       scanf("%d", &b[i]);
    }
    printf("Enter the size of the files :-\n");
    for (i = 1; i <= nf; i++)
    {
       printf("File %d:", i);
```

```
      scanf("%d", &f[i]);
    }
    for (i = 1; i <= nf; i++)
    {
      for (j = 1; j <= nb; j++)
      {
        if (bf[j] != 1)
        {
          temp = b[j] - f[i];
          if (temp >= 0)
            if (lowest > temp)
            {
              ff[i] = j;

              lowest = temp;
            }
        }
      }
      frag[i] = lowest;
      bf[ff[i]] = 1;
      lowest = 10000;
    }
    printf("\nFile No\tFile Size \tBlock No\tBlock Size\tFragment");
    for (i = 1; i <= nf && ff[i] != 0; i++)
      printf("\n%d\t\t%d\t\t\t%d\t\t\t%d\t\t\t%d", i, f[i], ff[i], b[ff[i]], frag[i]);
    break;

  case 3:
    printf("\n\tMemory Management Scheme - First Fit");
    printf("\nEnter the number of blocks:");
    scanf("%d", &nb);
    printf("Enter the number of files:");
    scanf("%d", &nf);
    printf("\nEnter the size of the blocks:-\n");
    for (i = 1; i <= nb; i++)
    {
      printf("Block %d:", i);
      scanf("%d", &b[i]);
    }
    printf("Enter the size of the files :-\n");
    for (i = 1; i <= nf; i++)
    {
      printf("File %d:", i);
      scanf("%d", &f[i]);
```

```
            }
            for (i = 1; i <= nf; i++)
            {
               for (j = 1; j <= nb; j++)
               {
                  if (bf[j] != 1)
                  {
                     temp = b[j] - f[i];
                     if (temp >= 0)
                     {
                        ff[i] = j;
                        break;
                     }
                  }
               }
               frag[i] = temp;
               bf[ff[i]] = 1;
            }
            printf("\nFile_no:\tFile_size :\tBlock_no:\tBlock_size:\tFragement");
            for (i = 1; i <= nf; i++)
               printf("\n%d\t\t%d\t\t\t%d\t\t\t%d\t\t\t%d", i, f[i], ff[i], b[ff[i]], frag[i]);
            break;

            case 4:
            printf("\nExiting...\n");
            break;
         default:
            printf("\nInvalid choice, please try again\n");
         }

      } while (choice != 4);

      return 0;
   }
```

Method-2:

```
   #include <stdio.h>
   #define max 25

   void first_fit(int b[], int f[], int nb, int nf, int bf[], int ff[], int frag[])
   {
      int i, j, temp;
      for (i = 1; i <= nf; i++)
      {
```

```
        for (j = 1; j <= nb; j++)
        {
            if (bf[j] != 1 && b[j] >= f[i])
            {
                ff[i] = j;
                frag[i] = b[j] - f[i];
                bf[j] = 1;
                break;
            }
        }
    }
    printf("\nWorst Fit:\nFile_no:\tFile_size :\tBlock_no:\tBlock_size:\tFragement");
    for (i = 1; i <= nf; i++)
        printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d", i, f[i], ff[i], b[ff[i]], frag[i]);
}

void best_fit(int b[], int f[], int nb, int nf, int bf[], int ff[], int frag[])
{
    int i, j, temp, lowest;
    for (i = 1; i <= nf; i++)
    {
        lowest = max;
        for (j = 1; j <= nb; j++)
        {
            if (bf[j] != 1 && b[j] >= f[i])
            {
                if (b[j] - f[i] < lowest)
                {
                    ff[i] = j;
                    lowest = b[j] - f[i];
                }
            }
        }
        frag[i] = lowest;
        bf[ff[i]] = 1;
    }
    printf("\nBest Fit:\nFile_no:\tFile_size :\tBlock_no:\tBlock_size:\tFragement");
    for (i = 1; i <= nf; i++)
        printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d", i, f[i], ff[i], b[ff[i]], frag[i]);
}

void worst_fit(int b[], int f[], int nb, int nf, int bf[], int ff[], int frag[])
{
    int i, j, temp, highest;
```

```
    for (i = 1; i <= nf; i++)
    {
        highest = 0;
        for (j = 1; j <= nb; j++)
        {
            if (bf[j] != 1 && b[j] >= f[i])
            {
                if (b[j] - f[i] > highest)
                {
                    ff[i] = j;
                    highest = b[j] - f[i];
                }
            }
        }
        frag[i] = highest;
        bf[ff[i]] = 1;
    }
    printf("\nFirst Fit:\nFile_no:\tFile_size :\tBlock_no:\tBlock_size:\tFragement");
    for (i = 1; i <= nf; i++)
        printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d", i, f[i], ff[i], b[ff[i]], frag[i]);
        printf("\n");
}

int main()
{
    int frag[max], b[max], f[max], i, nb, nf, temp;
    static int bf[max], ff[max];

    printf("\n\tMemory Management Scheme - Worst Fit, Best Fit, First Fit");
    printf("\nEnter the number of blocks:");
    scanf("%d", &nb);
    printf("Enter the number of files:");
    scanf("%d", &nf);
    printf("\nEnter the size of the blocks:\n");
    for (i = 1; i <= nb; i++)
    {
        printf("Block %d:", i);
        scanf("%d", &b[i]);
    }
    printf("Enter the size of the files:\n");
    for (i = 1; i <= nf; i++)
    {
        printf("File %d:", i);
        scanf("%d", &f[i]);
```

```
        }
        for (i = 1; i <= nb; i++)
            bf[i] = 0;
        for (i = 1; i <= nf; i++)
            ff[i] = 0;
        first_fit(b, f, nb, nf, bf, ff, frag);

        for (i = 1; i <= nb; i++)
            bf[i] = 0;
        for (i = 1; i <= nf; i++)
            ff[i] = 0;

        best_fit(b, f, nb, nf, bf, ff, frag);

        for (i = 1; i <= nb; i++)
            bf[i] = 0;
        for (i = 1; i <= nf; i++)
            ff[i] = 0;

        worst_fit(b, f, nb, nf, bf, ff, frag);

        return 0;
    }
```

==========

2. LRU Algorithm (Least Recently Used):

Method-1:

```
#include<stdio.h>

int main() {
    int frames, pages, i, j, k, l, count = 0, page_faults = 0;

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

    printf("Enter the number of pages: ");
    scanf("%d", &pages);

    int frame[frames], page[pages], flag[frames], counter[frames];

    for (i = 0; i < frames; i++) {
        frame[i] = -1;  // Initialize frames to -1 (indicating empty)
    }

    for (i = 0; i < pages; i++) {
```

```
    printf("Enter page number %d: ", i+1);
    scanf("%d", &page[i]);
}

printf("\nPage Replacement Process:\n");

for (i = 0; i < pages; i++) {
    int page_found = 0;

    // Check if page already exists in the frame
    for (j = 0; j < frames; j++) {
        if (frame[j] == page[i]) {
            page_found = 1;
            break;
        }
    }

    if (page_found == 0) {
        // Find the page with the least recently used counter
        for (j = 0; j < frames; j++) {
            flag[j] = 0;
            for (k = i - 1; k >= 0; k--) {
                if (frame[j] == page[k]) {
                    flag[j] = 1;
                    counter[j] = i - k;
                    break;
                }
            }
        }

        int min = 0;
        for (l = 1; l < frames; l++) {
            if (counter[l] < counter[min]) {
                min = l;
            }
        }

        // Replace the page with the least recently used page
        frame[min] = page[i];
        page_faults++;

        printf("\nPage fault occurred for page %d\n", page[i]);

        printf("Current Page Frames: ");
```

```
            for (j = 0; j < frames; j++) {
                printf("%d ", frame[j]);
            }
        }
    }

    printf("\n\nTotal page faults: %d\n", page_faults);

    return 0;
}
```

Method-2:

```
#include<stdio.h>

// Function to check if the page exists in the frame
int isPageInFrame(int page, int frame[], int frames) {
    for (int i = 0; i < frames; i++) {
        if (frame[i] == page) {
            return 1; // Page found in frame
        }
    }
    return 0; // Page not found in frame
}

// Function to find the index of the least recently used page
int findLRUIndex(int counter[], int frames) {
    int min = 0;
    for (int i = 1; i < frames; i++) {
        if (counter[i] < counter[min]) {
            min = i;
        }
    }
    return min;
}

// Function to simulate LRU page replacement algorithm
void simulateLRU(int pages[], int frames, int numOfPages) {
    int frame[frames];
    int counter[frames];
    int page_faults = 0;

    for (int i = 0; i < frames; i++) {
        frame[i] = -1; // Initialize frames to -1 (indicating empty)
```

```
        counter[i] = 0; // Initialize counters to 0
    }

    for (int i = 0; i < numOfPages; i++) {
        int page = pages[i];

        if (!isPageInFrame(page, frame, frames)) {
            int lruIndex = findLRUIndex(counter, frames);
            frame[lruIndex] = page;
            counter[lruIndex] = i + 1;
            page_faults++;
        } else {
            for (int j = 0; j < frames; j++) {
                if (frame[j] == page) {
                    counter[j] = i + 1;
                    break;
                }
            }
        }

        printf("\nPage %d:\n", page);
        printf("Current Page Frames: ");
        for (int j = 0; j < frames; j++) {
            printf("%d ", frame[j]);
        }
        printf("\n");
    }

    printf("\nTotal page faults: %d\n", page_faults);
}

int main() {
    int frames, numOfPages;

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

    printf("Enter the number of pages: ");
    scanf("%d", &numOfPages);

    int pages[numOfPages];

    printf("Enter the page numbers:\n");
    for (int i = 0; i < numOfPages; i++) {
```

```
        scanf("%d", &pages[i]);
    }

    printf("\nPage Replacement Process:\n");
    simulateLRU(pages, frames, numOfPages);

    return 0;
}
```

==========

3. Logical to physical memory address :

Method-1:

```
#include <stdio.h>
#define PAGE_SIZE 4

int main() {
    int logicalMemorySize, physicalMemorySize;
    int logicalAddress;
    int continueFlag = 0;

    printf("Enter the size of the logical memory: ");
    scanf("%d", &logicalMemorySize);

    printf("Enter the size of the physical memory: ");
    scanf("%d", &physicalMemorySize);

    int numPages = (logicalMemorySize + PAGE_SIZE - 1) / PAGE_SIZE;
    int numFrames = (physicalMemorySize + PAGE_SIZE - 1) / PAGE_SIZE;

    int pageTable[numPages];

    printf("\nEnter the page table:\n");
    printf("Page No\tFrame No\n-------\t-------\n");

    for (int i = 0; i < numPages; i++) {
        printf("%d\t\t", i);
        scanf("%d", &pageTable[i]);
    }

    do {
        printf("\nEnter the logical address: ");
        scanf("%d", &logicalAddress);
```

```
        int pageNumber = logicalAddress / PAGE_SIZE;
        int offset = logicalAddress % PAGE_SIZE;

        if (pageNumber < numPages && pageTable[pageNumber] != -1) {
            int frameNumber = pageTable[pageNumber];
            int physicalAddress = (frameNumber * PAGE_SIZE) + offset;
            printf("Physical address: %d\n", physicalAddress);
        } else {
            printf("Invalid logical address or page not present in any frame.\n");
        }

        printf("Do you want to continue (1/0)? ");
        scanf("%d", &continueFlag);
    } while (continueFlag == 1);

    return 0;
}
```

Method-2:

```
#include <stdio.h>

#define PAGE_SIZE 4

int main() {
    int logicalMemorySize, physicalMemorySize;
    int logicalAddress;
    int continueFlag = 0;

    printf("Enter the size of the logical memory: ");
    scanf("%d", &logicalMemorySize);

    printf("Enter the size of the physical memory: ");
    scanf("%d", &physicalMemorySize);

    int numPages = (logicalMemorySize + PAGE_SIZE - 1) / PAGE_SIZE;
    int numFrames = (physicalMemorySize + PAGE_SIZE - 1) / PAGE_SIZE;

    int pageTable[numPages];

    printf("\nEnter the page table:\n");
    printf("(Enter frame number as -1 if that page is not present in any frame)\n");
```

```
    for (int i = 0; i < numPages; i++) {
        printf("\nPage %d Frame: ", i);
        scanf("%d", &pageTable[i]);
    }

    do {
        printf("\nEnter the logical address: ");
        scanf("%d", &logicalAddress);

        int pageNumber = logicalAddress / PAGE_SIZE;
        int offset = logicalAddress % PAGE_SIZE;

        if (pageNumber < numPages && pageTable[pageNumber] != -1) {
            int frameNumber = pageTable[pageNumber];
            int physicalAddress = (frameNumber * PAGE_SIZE) + offset;
            printf("Physical address: %d\n", physicalAddress);
        } else {
            printf("Invalid logical address or page not present in any frame.\n");
        }

        printf("Do you want to continue (1/0)? ");
        scanf("%d", &continueFlag);
    } while (continueFlag == 1);

    return 0;
}
```

Method-3:

```
#include <stdio.h>
#include <math.h>

#define PAGE_SIZE 4

int main() {
    int logicalMemorySize, physicalMemorySize;
    int logicalAddress;
    int continueFlag = 0;

    printf("Enter the size of the logical memory: ");
    scanf("%d", &logicalMemorySize);

    printf("Enter the size of the physical memory: ");
    scanf("%d", &physicalMemorySize);
```

```
    int numPages = ceil((double)logicalMemorySize / PAGE_SIZE);
    int numFrames = ceil((double)physicalMemorySize / PAGE_SIZE);

    int pageTable[numPages];

    printf("\nEnter the page table:\n");
    printf("(Enter frame number as -1 if that page is not present in any frame)\n");

    for (int i = 0; i < numPages; i++) {
       printf("\nPage %d Frame: ", i);
       scanf("%d", &pageTable[i]);
    }

    do {
       printf("\nEnter the logical address: ");
       scanf("%d", &logicalAddress);

       int pageNumber = logicalAddress / PAGE_SIZE;
       int offset = logicalAddress % PAGE_SIZE;

       if (pageNumber < numPages && pageTable[pageNumber] != -1) {
          int frameNumber = pageTable[pageNumber];
          int physicalAddress = (frameNumber * PAGE_SIZE) + offset;
          printf("Physical address: %d\n", physicalAddress);
       } else {
          printf("Invalid logical address or page not present in any frame.\n");
       }

       printf("Do you want to continue (1/0)? ");
       scanf("%d", &continueFlag);
    } while (continueFlag == 1);

    return 0;
}
```

==========

4. Optimal Page Replacement Algorithm:

Method-1:

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

#define FRAME_SIZE 3
```

```
int main()
{
   int page_reference_string[] = {1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 5};
   int page_reference_length = sizeof(page_reference_string) /
sizeof(page_reference_string[0]);
   int page_faults = 0;
   int page_hits = 0;
   int frames[FRAME_SIZE];
   int frame_index = 0;
   int page_index = 0;
   int i, j, k;

   for (i = 0; i < FRAME_SIZE; i++) {
      frames[i] = -1;
   }

   printf("Page reference string: ");
   for (i = 0; i < page_reference_length; i++) {
      printf("%d ", page_reference_string[i]);
   }
   printf("\n\n");

   printf("Frames\tPage Faults\tPage Hits\n");
   printf("------\t-----------\t---------\n");

   for (i = 0; i < page_reference_length; i++) {
      int page_number = page_reference_string[i];
      int is_page_hit = 0;

      for (j = 0; j < FRAME_SIZE; j++) {
         if (frames[j] == page_number) {
            is_page_hit = 1;
            page_hits++;
            break;
         }
      }

      if (!is_page_hit) {
         int is_frame_empty = 0;

         for (j = 0; j < FRAME_SIZE; j++) {
            if (frames[j] == -1) {
               frames[j] = page_number;
```

```
            is_frame_empty = 1;
            break;
        }
    }

    if (!is_frame_empty) {
        int max_future_distance = -1;
        int page_to_replace_index = -1;

        for (j = 0; j < FRAME_SIZE; j++) {
            int page_in_frame = frames[j];
            int future_distance = 0;

            for (k = i + 1; k < page_reference_length; k++) {
                if (page_reference_string[k] == page_in_frame) {
                    break;
                }
                future_distance++;
            }

            if (future_distance > max_future_distance) {
                max_future_distance = future_distance;
                page_to_replace_index = j;
            }
        }

        frames[page_to_replace_index] = page_number;
    }

    page_faults++;
  }

  printf(" ");
  for (j = 0; j < FRAME_SIZE; j++) {
    if (frames[j] == -1) {
        printf("- ");
    } else {
        printf("%d ", frames[j]);
    }
  }
  printf("\t\t%d\t\t%d\n", page_faults, page_hits);
}

return 0;
```

```
}
```

Method-2:

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

#define FRAME_SIZE 3 // Define the number of frames

int main()
{
   int page_reference_string[] = {1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 5, 6, 7, 1, 2, 3, 4, 5, 6,
7, 8, 1, 2, 3, 4, 5, 6, 7, 8, 9}; // Initialize an array of page reference string
   int page_reference_length = sizeof(page_reference_string) /
sizeof(page_reference_string[0]); // Calculate the length of the page reference string
   int page_faults = 0; // Initialize the number of page faults
   int page_hits = 0; // Initialize the number of page hits
   int frames[FRAME_SIZE]; // Initialize an array of frames
   int frame_index = 0; // Initialize the index of the current frame
   int page_index = 0; // Initialize the index of the current page
   int i, j, k; // Initialize loop variables

   for (i = 0; i < FRAME_SIZE; i++) { // Initialize all frames to -1, indicating that they are all
empty
      frames[i] = -1;
   }

   printf("Page reference string: ");
   for (i = 0; i < page_reference_length; i++) { // Print the page reference string
      printf("%d ", page_reference_string[i]);
   }
   printf("\n\n");

   printf("Frames\tPage Faults\tPage Hits\n");
   printf("------\t-----------\t---------\n");

   for (i = 0; i < page_reference_length; i++) { // Iterate through the page reference string
      int page_number = page_reference_string[i]; // Get the current page number
      int is_page_hit = 0; // Initialize the flag for whether the page is a hit or not

      for (j = 0; j < FRAME_SIZE; j++) { // Check if the page is already in one of the frames
         if (frames[j] == page_number) {
            is_page_hit = 1; // Set the flag if the page is a hit
```

```
            page_hits++; // Increment the number of page hits
            break;
        }
    }

    if (!is_page_hit) { // If the page is not a hit, it is a page fault
        int is_frame_empty = 0; // Initialize the flag for whether a frame is empty or not

        for (j = 0; j < FRAME_SIZE; j++) { // Check if there is an empty frame
            if (frames[j] == -1) {
                frames[j] = page_number; // If there is an empty frame, add the page to it
                is_frame_empty = 1; // Set the flag to indicate that an empty frame was found
                break;
            }
        }

        if (!is_frame_empty) { // If there are no empty frames, find the page that will not be used for the longest period of time in the future and replace it with the current page
            int max_future_distance = -1;
            int page_to_replace_index = -1;

            for (j = 0; j < FRAME_SIZE; j++) {
                int page_in_frame = frames[j];
                int future_distance = 0;

                for (k = i + 1; k < page_reference_length; k++) {
                    if (page_reference_string[k] == page_in_frame) {
                        break;
                    }
                    future_distance++;
                }

                if (future_distance > max_future_distance) {
                    max_future_distance = future_distance;
                    page_to_replace_index = j;
                }
            }

            frames[page_to_replace_index] = page_number;
        }

        page_faults++;
    }
```

```
        printf(" ");
        for (j = 0; j < FRAME_SIZE; j++) {
            if (frames[j] == -1) {
                printf("- ");
            } else {
                printf("%d ", frames[j]);
            }
        }
        printf("\t\t%d\t\t%d\n", page_faults, page_hits);
    }

    return 0;
}
```