**Assignment 8**

**Name** : Komal Potdar

**Roll No**.: 92

**PRN No**.: 12320165

**Div**: CS B SY

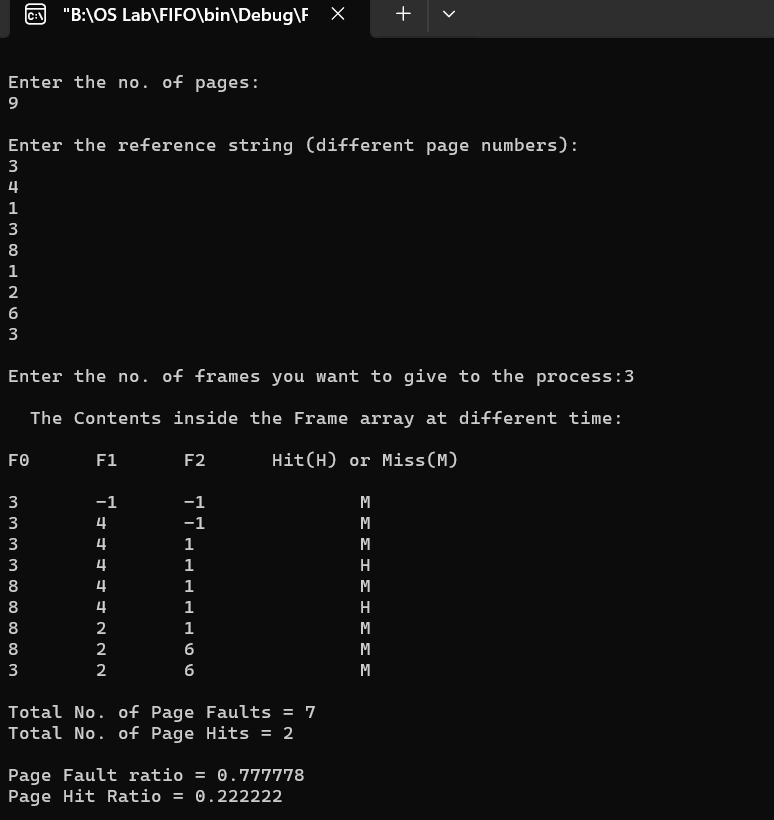
**Batch**: 3

**Implement Page replacement Algorithms**

1. **FIFO**

|  |
| --- |
| #include <stdio.h>  #include <stdbool.h>  struct PageTable  {      int frame\_no;      bool valid;  };  bool isPagePresent(struct PageTable PT[], int page, int n)  {      if (PT[page].valid == true)      {          return true;      }      return false;  }  void updatePageTable(struct PageTable PT[], int page, int fr\_no, int status)  {      PT[page].valid = status;      PT[page].frame\_no = fr\_no;  }  void printFrameContents(int frame[], int no\_of\_frames)  {      for (int i = 0; i < no\_of\_frames; i++)          printf("%d\t", frame[i]);      printf("\tM\n");  }  void printFrameContentsH(int frame[], int no\_of\_frames)  {      for (int i = 0; i < no\_of\_frames; i++)          printf("%d\t", frame[i]);      printf("\tH\n");  }  int main()  {      int n, no\_of\_frames, page\_fault = 0, page\_hits = 0, current = 0;      bool flag = false;      printf("\nEnter the no. of pages:\n");      scanf("%d", &n);      int reference\_string[n];      printf("\nEnter the reference string (different page numbers):\n");      for (int i = 0; i < n; i++)      {          scanf("%d", &reference\_string[i]);      }      printf("\nEnter the no. of frames you want to give to the process:");      scanf("%d", &no\_of\_frames);      int frame[no\_of\_frames];      for (int i = 0; i < no\_of\_frames; i++)      {          frame[i] = -1;      }      struct PageTable PT[50];      for (int i = 0; i < 50; i++)      {          PT[i].valid = false;      }      printf("\n  The Contents inside the Frame array at different time:  \n\n");      for (int i = 0; i < no\_of\_frames; i++)      {          printf("F%d\t", i);      }      printf("Hit(H) or Miss(M)\n\n");      for (int i = 0; i < n; i++)      {          if (!(isPagePresent(PT, reference\_string[i], n)))          {              page\_fault++;              if (flag == false && current < no\_of\_frames)              {                  frame[current] = reference\_string[i];                  printFrameContents(frame, no\_of\_frames);                  updatePageTable(PT, reference\_string[i], current, true);                  current = current + 1;                  if (current == no\_of\_frames)                  {                      current = 0;                      flag = true;                  }              }              else              {                  updatePageTable(PT, frame[current], -1, false);                  frame[current] = reference\_string[i];                  printFrameContents(frame, no\_of\_frames);                  updatePageTable(PT, reference\_string[i], current, true);                  current = (current + 1) % no\_of\_frames;              }          }          else          {              printFrameContentsH(frame, no\_of\_frames);              page\_hits++;          }      }      printf("\nTotal No. of Page Faults = %d\n", page\_fault);      printf("Total No. of Page Hits = %d\n", page\_hits);      printf("\nPage Fault ratio = %f\n", (float)page\_fault / n);      printf("Page Hit Ratio = %f\n", (float)page\_hits / n);      return 0;  } |

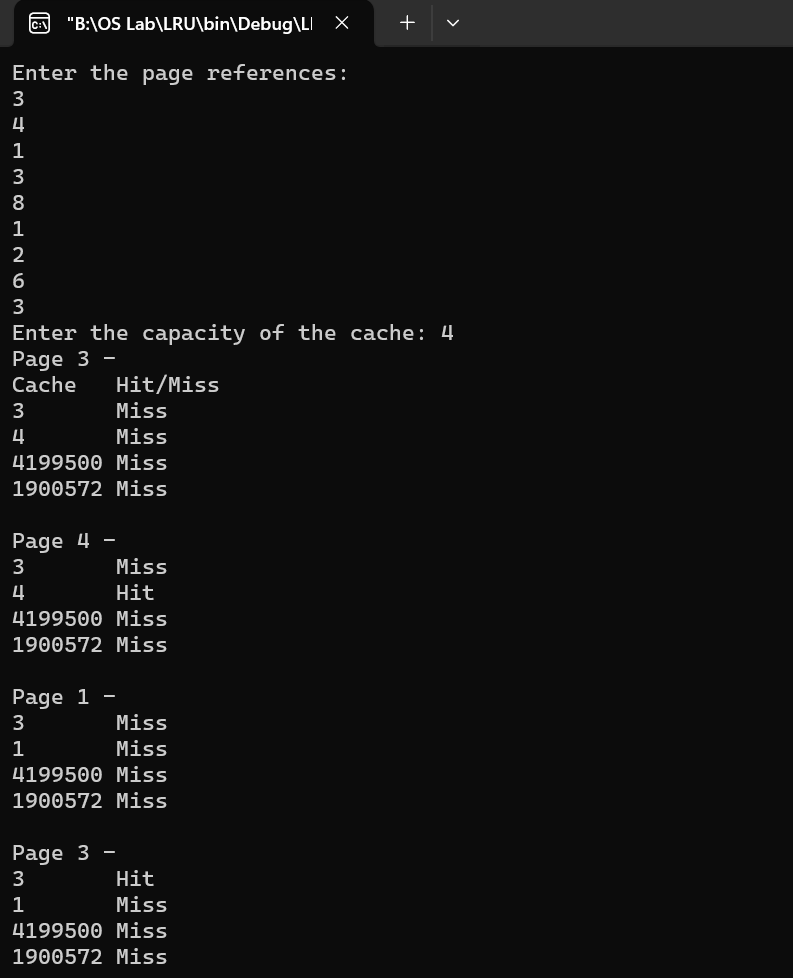
**Output:**

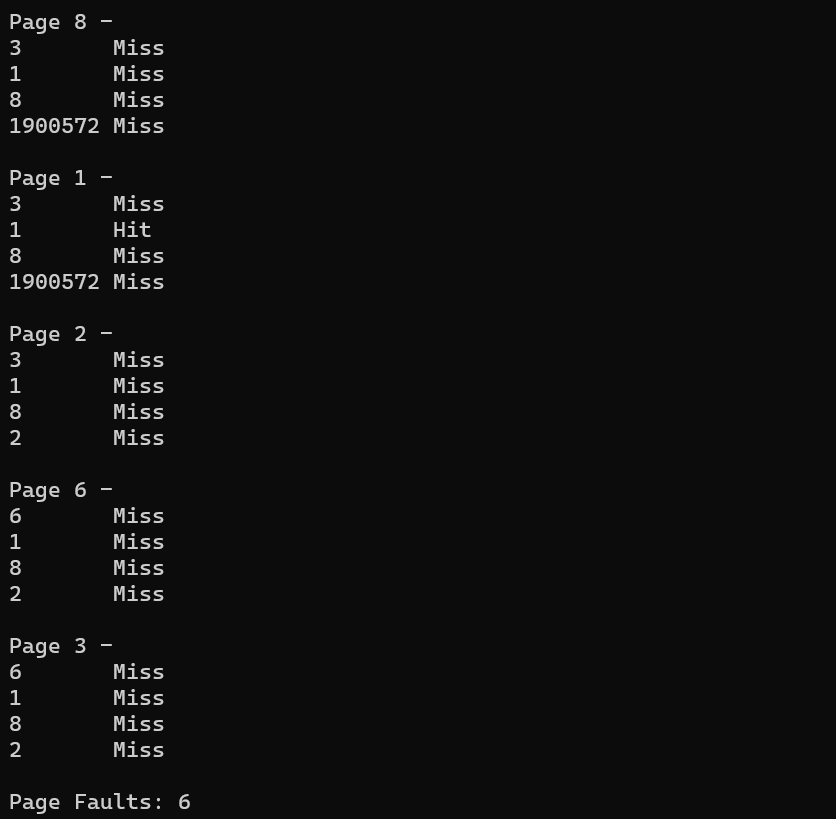
****

1. **LRU**

|  |
| --- |
| #include <stdio.h>  #include <stdbool.h>  #include <limits.h>  void displayCacheHeader()  {      printf("Cache\tHit/Miss\n");  }  void displayCache(int s[], int n, int page, bool isHit, bool displayHeader)  {      if (displayHeader)      {          displayCacheHeader();      }      for (int i = 0; i < n; i++)      {          printf("%d\t", s[i]);          if (s[i] == page && isHit)          {              printf("Hit");          }          else          {              printf("Miss");          }          printf("\n");      }      printf("\n");  }  int pageFaults(int pages[], int n, int capacity)  {      int s[capacity];      int indexes[1000] = {0};      int page\_faults = 0;      bool displayHeader = true;      for (int i = 0; i < n; i++)      {          printf("Page %d - \n", pages[i]);          bool isHit = false;          for (int j = 0; j < capacity; j++)          {              if (s[j] == pages[i])              {                  isHit = true;                  break;              }          }          if (!isHit)          {              if (page\_faults == capacity)              {                  int lru = INT\_MAX, val;                  for (int j = 0; j < capacity; j++)                  {                      if (indexes[s[j]] < lru)                      {                          lru = indexes[s[j]];                          val = s[j];                      }                  }                  for (int j = 0; j < capacity; j++)                  {                      if (s[j] == val)                      {                          s[j] = pages[i];                          break;                      }                  }              }              else              {                  s[page\_faults] = pages[i];              }              page\_faults++;          }          displayCache(s, capacity, pages[i], isHit, displayHeader);          displayHeader = false;          indexes[pages[i]] = i;      }      return page\_faults;  }  int main()  {      int n;      printf("Enter the number of pages: ");      scanf("%d", &n);      int pages[n];      printf("Enter the page references:\n");      for (int i = 0; i < n; i++)      {          scanf("%d", &pages[i]);      }      int capacity;      printf("Enter the capacity of the cache: ");      scanf("%d", &capacity);      printf("Page Faults: %d\n", pageFaults(pages, n, capacity));      return 0;  } |

**Output:**

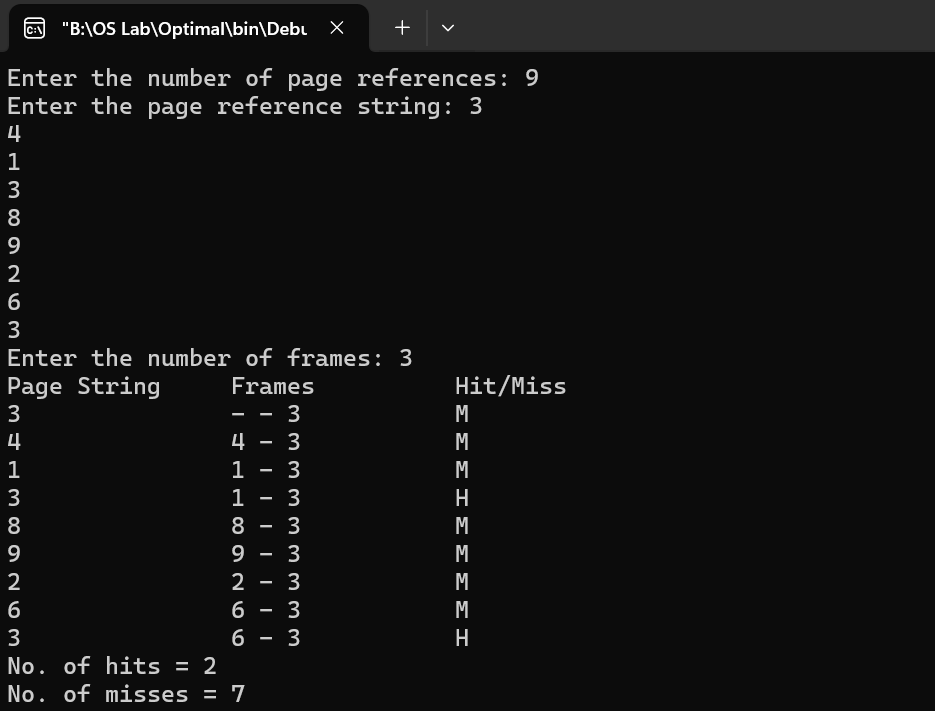
****

****

1. **Optimal**

|  |
| --- |
| #include <stdio.h>  #include <stdbool.h>  bool search(int key, int fr[], int fr\_size)  {      for (int i = 0; i < fr\_size; i++)      {          if (fr[i] == key)          {              return true;          }      }      return false;  }  int predict(int pg[], int pn, int fr[], int fr\_size, int index)  {      int res = -1, farthest = index;      for (int i = 0; i < fr\_size; i++)      {          int j;          for (j = index; j < pn; j++)          {              if (fr[i] == pg[j])              {                  if (j > farthest)                  {                      farthest = j;                      res = i;                  }                  break;              }          }          if (j == pn)          {              return i;          }      }      return (res == -1) ? 0 : res;  }  void printFrames(int fr[], int fr\_size)  {      for (int i = 0; i < fr\_size; i++)      {          if (fr[i] == -1)          {              printf("- ");          }          else          {              printf("%d ", fr[i]);          }      }  }  void optimalPage(int pg[], int pn, int fn)  {      int fr[fn];      for (int i = 0; i < fn; i++)      {          fr[i] = -1;      }      int hit = 0;      char result;      printf("Page String\tFrames\t\tHit/Miss\n");      for (int i = 0; i < pn; i++)      {          printf("%d\t\t", pg[i]);          if (search(pg[i], fr, fn))          {              hit++;              result = 'H';          }          else          {              if (fr[fn - 1] == -1)              {                  fr[fn - 1] = pg[i];                  result = 'M';              }              else              {                  int j = predict(pg, pn, fr, fn, i + 1);                  result = 'M';                  fr[j] = pg[i];              }          }          printFrames(fr, fn);          printf("\t\t%c\n", result);      }      printf("No. of hits = %d\n", hit);      printf("No. of misses = %d\n", pn - hit);  }  int main()  {      int pn, fn;      printf("Enter the number of page references: ");      scanf("%d", &pn);      int pg[pn];      printf("Enter the page reference string: ");      for (int i = 0; i < pn; i++)      {          scanf("%d", &pg[i]);      }      printf("Enter the number of frames: ");      scanf("%d", &fn);      optimalPage(pg, pn, fn);      return 0;  } |

**Output:**

****