First In First Out (FIFO)

```
#include<bits/stdc++.h>
using namespace std;
int main(){
int n,m,i,j,k,hit=0;
cout<<"Enter number of frames\n";
cin>>n;
cout<<"Enter number of processes\n";
cin>>m;
vector<int> p(m);
vector<int> hi(m);
cout<<"Enter processes\n";
for(i=0;i<m;i++){
cin>>p[i];
}
vector<vector<int>> a(n);
for(i=0;i<n;i++){
a[i]=vector<int>(m,-1);
}
map <int, int> mp;
for(i=0;i<m;i++){
vector<pair<int,int>> c;
for(auto q: mp){
c.push_back({q.second,q.first});
}
sort(c.begin(),c.end());
bool hasrun=false;
for(j=0;j<n;j++){
if(a[j][i]==p[i]){
hit++;
hi[i]=1;
mp[p[i]]++;
hasrun=true;
break;
}
if(a[j][i]==-1){
for(k=i;k<m;k++)
a[j][k]=p[i];
mp[p[i]]++;
hasrun=true;
break;
}
}
if(j==n||hasrun==false){
for(j=0;j<n;j++){
if(a[j][i]==c[c.size()-1].second){
mp.erase(a[j][i]);
for(k=i;k<m;k++)
a[j][k]=p[i];
```

```
mp[p[i]]++;
break;
}
}
}
for(auto q:mp){
if(q.first!=p[i]){
mp[q.first]++;
}
}
}
cout<<"Process ";
for(i=0;i<m;i++){
cout<<p[i]<<" ";
}
cout<<'\n';
for(i=0;i<n;i++){
cout<<"Frame "<<i<<" ";
for(j=0;j<m;j++){
if(a[i][j]==-1)
cout<<"E ";
else
cout<<a[i][j]<<" ";
}
cout<<'\n';
}
for(i=0;i<m;i++){
if(hi[i]==0)
cout<<" ";
else
cout<<hi[i]<<" ";
}
cout<<"\n";
cout<<"Hit "<<hit<<'\n'<<"Page Fault "<<m-hit<<'\n';
return 0;
}
```

Output:

```
Enter number of frames
3
Enter number of processes
12
Enter processes
1 2 3 4 1 2 5 1 2 3 4 5
Process 1 2 3 4 1 2 5 1 2 3 4 5
Frame 0 1 1 1 4 4 4 5 5 5 5 5 5
Frame 1 E 2 2 2 1 1 1 1 1 3 3 3
Frame 2 E E 3 3 3 2 2 2 2 2 4 4
        1 1   1
Hit 3
Page Fault 9
```

Least Recently Used (LRU) Page Replacement Algorithm

```
#include<bits/stdc++.h>
using namespace std;
int main(){
   int n,m,i,j,k,hit=0;
   cout<<"Enter number of frames\n";
   cin>>n;
   cout<<"Enter number of processes\n";
   cin>>m;
   vector<int> p(m);
   vector<int> hi(m);
   cout<<"Enter processes\n";
   for(i=0;i<m;i++){
      cin>>p[i];
   }
   vector<vector<int>> a(n);
   for(i=0;i<n;i++){
      a[i]=vector<int>(m,-1);
   }
   map <int, int> mp;
   for(i=0;i<m;i++){
      vector<pair<int,int>> c;
      for(auto q: mp){
         c.push_back({q.second,q.first});
      }
      sort(c.begin(),c.end());
      bool hasrun=false;
      for(j=0;j<n;j++){
         if(a[j][i]==p[i]){
            hit++;
            hi[i]=1;
            mp[p[i]]=1;
            hasrun=true;
            break;
         }
         if(a[j][i]==-1){
            for(k=i;k<m;k++)
               a[j][k]=p[i];
            mp[p[i]]++;
            hasrun=true;
            break;
         }
      }
      if(j==n||hasrun==false){
         for(j=0;j<n;j++){
            if(a[j][i]==c[c.size()-1].second){
               mp.erase(a[j][i]);
               for(k=i;k<m;k++)
                  a[j][k]=p[i];
               mp[p[i]]++;
```

```
                break;
            }
        }
    }
    for(auto q:mp){
        if(q.first!=p[i]){
            mp[q.first]++;
        }
    }
  }
  cout<<"Process ";
  for(i=0;i<m;i++){
cout<<p[i]<<" ";
  }
  cout<<'\n';
  for(i=0;i<n;i++){
    cout<<"Frame "<<i<<" ";
    for(j=0;j<m;j++){
        if(a[i][j]==-1)
            cout<<"E ";
            else
        cout<<a[i][j]<<" ";
    }
    cout<<'\n';
  }
  for(i=0;i<m;i++){
    if(hi[i]==0)
    cout<<"  ";
    else
    cout<<hi[i]<<" ";
  }
  cout<<"\n";
  cout<<"Hit "<<hit<<'\n'<<"Page Fault "<<m-hit<<'\n';
  return 0;
}
```

Output;

```
Enter number of frames
3
Enter number of processes
12
Enter processes
1 2 3 4 1 2 5 1 2 3 4 5
Process 1 2 3 4 1 2 5 1 2 3 4 5
Frame 0 1 1 1 4 4 4 5 5 5 3 3 3
Frame 1 E 2 2 2 1 1 1 1 1 1 4 4
Frame 2 E E 3 3 3 2 2 2 2 2 2 5
                1 1
Hit 2
Page Fault 10
```

Optimal Page Replacement Algorithm

```
#include <bits/stdc++.h>
using namespace std;
int predict(int page[], vector<int>& fr, int pn, int index) {
  // Store the index of pages which are going
  // to be used recently in future
  int res = -1, farthest = index;
  for (int i = 0; i < fr.size(); i++) {
    int j;
    for (j = index; j < pn; j++) {
      if (fr[i] == page[j]) {
        if (j > farthest) {
          farthest = j;
          res = i;
        }
        break;
      }
    }
    // Return the page which are
    // are never referenced in future,
    if (j == pn)
      return i;
  }
  // If all of the frames were not in future,
  // return any of them, we return 0. Otherwise
  // we return res.
  return (res == -1) ? 0 : res;
}
bool search(int key, vector<int>& fr) {
  for (int i = 0; i < fr.size(); i++)
  if (fr[i] == key)
  return true;
  return false;
}
void opr(int page[], int pn, int fn) {
  vector<int> fr;
  int hit = 0;
  for (int i = 0; i < pn; i++) {
    // Page found in a frame : HIT
    if (search(page[i], fr)) {
      hit++;
      continue;
    }
    //If a page not found in a frame : MISS
    // check if there is space available in frames.
    if (fr.size() < fn)
    fr.push_back(page[i]);
```

```
      // Find the page to be replaced.
      else {
        int j = predict(page, fr, pn, i + 1);
        fr[j] = page[i];
      }
    }
    cout << "Hits = " << hit << endl;
    cout << "Misses = " << pn - hit << endl;
}
// main Function
int main() {
    int page[] = { 1, 7, 8, 3, 0, 2, 0, 3, 5, 4, 0, 6, 1 };
    int pn = sizeof(page) / sizeof(page[0]);
    int fn = 3;
    opr(page, pn, fn);
    return 0;
}
```

Output:

```
Hits = 3
Misses = 10
```