```
// Vahe Karamian - Project III - CS 431 - Dr. Lee
// Filename: taskTable.h

#include <iostream>
#include <iomanip>
#include <string>
#include <math.h>

using namespace std;

class TaskTable
{
  public:
    TaskTable();

    // Update process id, process size, and process state in the
    // Task Table
    void Update( string processID, int size, string state);

    // Update alpha
    void Update( int alpha );

    // Update state
    void Update( string state );

    string getState( )
    {
      return state;     // return process state
    }

    string getID( )
    {
      return ID;        // return process ID
    }

    int getAlpha( )
    {
      return alpha;     // return process alpha
    }

    int getSize( )
    {
      return size;      // return process size
    }

    int getPages( )
    {
      return pnum;      // return process pages number required
    }

  private:
    string state; // process state in the task table
    string ID;    // process id in the task table
    int alpha;    // process alpha in the task table
    int size;     // process size in the task table
    int pnum;     // process page numbers required

    void round( )
    {
      int c=1;
      int temp = size;
      while( temp >500 )
      {
        temp-=500;
        c++;
      }
      pnum = c;
    }
};

TaskTable::TaskTable()
{
  ID = " ";       // set process id
  alpha = 0;      // set process alpha
  size = 0;       // set process size
  state = " ";    // set process state
  pnum = 0;

  // round( );        // calculate number of pages required
```

```
}

void TaskTable::Update(string id, int len, string s)
{
  ID = id;        // set process id
  size = len;     // set process size
  state = s;      // set process state

  round( );       // calculate number of pages required
}

void TaskTable::Update(int p)
{
  alpha = p;      // set alpha
}

void TaskTable::Update(string s)
{
  state = s;      // set the state
}

// structure for Page Map Table
struct PMT
{
  string pid;     // process id
  int fn;         // frame number stored at
};

// structure for Main Memory
struct MM
{
  string pid;     // process id
  int pn;         // page number in PMT
};



// Vahe Karamian - Project III - CS 431 - Dr. Lee
// Filename: project3.cpp

#include "taskTable.h"

#define     MAX 32                    // define maximum

// function declarations
void  initialize( );                  // initialize function
int   menu( );                        // display a menu
int   waitingProcesses( int index );  // WP taken care of
void  printDS( int index );           // print data structures

// global variables
PMT         *MPMT[32];                // Master Page Map Table
                                      // keep pointer value to allocated
                                      // Page Map Table
MM          mainMemory[32];           // Main Memory array

TaskTable TB[32];                     // Task Table array

int         GLOBAL_COUNTER = 0;       // Global Counter used for
                                      // Master Page Map Table
int         WAITING_COUNTER = 0;      // Waiting Counter used for
                                      // waiting processes

void main()
{
  int selection = 0;                  // used for menu selection
  int index = 0;                      // used for Task Table management

  initialize();                       // initialize all data structures

  while( selection != 2 )             // while user doesn't want to quit
  {
    selection = menu( );              // get user request

    switch( selection )               // process user request
    {
      case 0:                         // start a process
        {
          int counter = 0;            // used for free frames in memory
```

```
  int pageNumber = 0;         // used for calculating pages required
  int s;                      // used for size of the process
  bool FLAG = false;          // logical FLAG
  string id;                  // used for process id

  cout<<"Enter id: ";         // get process id
  cin>>id;

  cout<<"Enter the size: ";   // get process size
  cin>>s;


  // Update the Task Table, process id, size, and state
  TB[index].Update( id, s, "ready" );

  // calculate number of pages required
  // for this process
  pageNumber = TB[index].getPages();

  // Go through the main memory
  for(int i = 0; i < 32; i++)
  {
    // if memory is available then increase the counter
    if(mainMemory[i].pid=="###")
      counter++;
    // if counter >= # of pages required set FLAG and break
    if( counter >= pageNumber ) { FLAG = true; break;}
  }

  // if FLAG is set - memory available for process
  if( FLAG )
  {
    // allocate a Page Map Table of size # of pages
    PMT * process = new PMT[pageNumber];

    // update Page Map Table and Main Memory to correspond
    for( int k=0; k<pageNumber; k++ )
    {
      for( int j=0; j<32; j++ )
      {
        if( mainMemory[j].pid == "###" )
        {
          mainMemory[j].pid = id; // Main Memory process id at location j
          mainMemory[j].pn = k;   // Main Memory page number at location j
          process[k].pid = id;    // PMT process id at location k
          process[k].fn = j;      // PMT frame number at location k
          break;
        }
      }
    }

    // while Master Page Map Table != NULL
    while( MPMT[GLOBAL_COUNTER] != NULL )
    {
      GLOBAL_COUNTER++;         // increment the Global Counter
      if( GLOBAL_COUNTER>31 )   // if Global Counter > 31
        GLOBAL_COUNTER = 0;     // let Global Counter = 0
    }

    // insert process pointer into the Master Page Map Table
    MPMT[GLOBAL_COUNTER] = process;

    TB[index].Update("running");      // update process state in Task Table
    TB[index].Update(GLOBAL_COUNTER); // update process alpha in Task Table
    index++;                          // increment index
  }

  // if FLAG is not set
  if(!FLAG)
  {
    WAITING_COUNTER++;        // increment Waiting Counter
    index++;                  // increment index
  }

  printDS( index );

  break;                      // display menu
}
```

```
    case 1:                         // terminate a process
      {
        int a    = 0;               // alpha variable
        int size = 0;               // size variable
        int pageNumber = 0;         // # pages variable

        string id;                  // process id variable

        cout<<"Enter id: ";         // get process id
        cin>>id;

        // get process alpha from the Task Table
        for(int i = 0; i <index; i++)
        {
          // if there is a match get the alpha from the Task Table
          // update Task Table so that process is halted and break
          if( TB[i].getID() == id )
          {
            a = TB[i].getAlpha( );
            TB[i].Update( "halt" );
            pageNumber = TB[i].getPages( ); // get number of pages required
            break;
          }
        }

        PMT *ptr;         // create a Page Map Table pointer
        ptr = MPMT[a];    // point it to Master Page Map Table at position alpha

        // remove process from Main Memory
        for(i=0; i<pageNumber; i++)
        {
          mainMemory[ ptr[i].fn ].pid="###";
          mainMemory[ ptr[i].fn ].pn=0;
        }

        // remove pointer from the Master Page Map Table
        MPMT[a] = NULL;

        // deallocate the allocated Page Map Table for this specific process
        delete [] ptr;

        int c = WAITING_COUNTER;

        // Check to see if we have anything in the Waiting Queue - if so
        if( WAITING_COUNTER )
        {
          // while we have processes waiting
          while( 0 <= c )
          {
            // check the Task Table
            for( i=0; i<=index; i++ )
            {
              if(TB[i].getState( ) == "ready")
              {
                // decrement the Waiting Counter
                WAITING_COUNTER--;

                // if WaitingProcess did not get into Memory
                if( waitingProcesses( i ) == 1 )
                  WAITING_COUNTER++;  // increment Waiting Counter
                break;                // break
              }
            }
            c--;  // decrement c
          }
        }

        printDS( index );

        break;                      // display the menu
      }

    default:
      {
        // error
        break;
      }
  }
}
```

```
  printDS( index );
}

// initialize all of the data structures
void initialize()
{
  for(int i = 0; i < 32; i++)
  {
    mainMemory[i].pid = "###";
    mainMemory[i].pn = 0;
    MPMT[i] = NULL;
  }
}

// menu function for the program
int menu()
{
  int selection;
  cout<<"0. Start"<<endl
      <<"1. Terminate"<<endl
      <<"2. Quit"<<endl<<endl
      <<"> ";
  cin>>selection;
  return selection;
}

// process any waiting processes in the waiting queue
int waitingProcesses( int index )
{
  int counter = 0;                    // used for free frames in memory

  int pageNumber = 0;                 // used for calculating pages required
  bool FLAG = false;                  // logical FLAG

  // get number of pages required
  pageNumber = TB[index].getPages( );

  // Go through the main memory
  for(int i = 0; i < 32; i++)
  {
    // if memory is available then increase the counter
    if(mainMemory[i].pid=="###")
    {
      counter++;  //size available
    }
    // if counter >= # of pages required set FLAG and break
    if(counter >= pageNumber) { FLAG = true; break;}
  }

  // if FLAG is set - memory available for process
  if(FLAG)
  {
    // allocate a Page Map Table of size # of pages
    PMT * process = new PMT[pageNumber];

    // update Page Map Table and Main Memory to correcspond
    for( int k=0; k<pageNumber; k++ )
    {
      for( int j=0; j<32; j++ )
      {
        if( mainMemory[j].pid == "###" )
        {
          // Main Memory process id at location j
          mainMemory[j].pid = TB[index].getID();
          // Main Memory page number at location j
          mainMemory[j].pn = k;
          // PMT process id at location k
          process[k].pid = TB[index].getID();
          // PMT frame number at location k
          process[k].fn = j;
          break;
        }
      }
    }

    // while Master Page Map Table != NULL
    while( MPMT[GLOBAL_COUNTER] != NULL )
    {
      GLOBAL_COUNTER++;         // increment the Global Counter
```

```
      if( GLOBAL_COUNTER>31 )   // if Global Counter > 31
        GLOBAL_COUNTER = 0;     // let Global Counter = 0
    }

    // insert process pointer into the Master Page Map Table
    MPMT[GLOBAL_COUNTER] = process;

    TB[index].Update("running");      // update process state in Task Table
    TB[index].Update(GLOBAL_COUNTER); // update process alpha in Task Table
  }

  // if FLAG is not set
  if(!FLAG)
    return(1);  // No memory available to allocate!

  return(0);
}

// print function to print all the data structures
void printDS( int index )
{
  cout<<endl<<endl;

  // Printing the Main Memory
  cout<<setw(8)<<"PID"<<setw(10)<<"PN"<<endl;
  cout<<setw(8)<<"==="<<setw(10)<<"=="<<endl;

  for( int i=0; i<32; i++ )
  {
    cout<<setw(8)
        <<mainMemory[i].pid
        <<setw(10)
        <<mainMemory[i].pn
        <<endl;
  }

  cout<<endl<<endl;

  // Printing the Task Table
  cout<<setw(8)<<"PID"<<setw(10)<<"SIZE"<<setw(10)<<"PAGES"<<setw(10)<<"STATE"<<endl;
  cout<<setw(8)<<"==="<<setw(10)<<"===="<<setw(10)<<"====="<<setw(10)<<"====="<<endl;

  for( i=0; i<index; i++ )
    cout<<setw(8)
        <<TB[i].getID()
        <<setw(10)
        <<TB[i].getSize()
        <<setw(10)
        <<TB[i].getPages()
        <<setw(10)
        <<TB[i].getState()<<endl;

  cout<<endl<<endl;

  // Printing the Master Page Map Table and Page Map Table Content
  cout<<setw(8)<<"PID"<<setw(15)<<"MPMT->PMT[]"<<endl;
  cout<<setw(8)<<"==="<<setw(15)<<"==========="<<endl;

  for( i=0; i<=index; i++ )
  {
    if( MPMT[i] != NULL )
    {
      cout<<setw(8)
          <<MPMT[i]->pid
          <<setw(15)
          <<MPMT[i]->fn;

      PMT *temp = MPMT[i];

      int pageNumber=0;

      // get process alpha from the Task Table
      for(int z = 0; z <index; z++)
      {
        // if there is a match get the alpha from the Task Table
        // update Task Table so that process is halted and break
        if( TB[z].getID() == MPMT[i]->pid )
        {
          pageNumber = TB[z].getPages( ); // get number of pages required
```

```
          break;
        }
      }

      cout<<"\t";

      for( int v=0; v<pageNumber; v++ )
      {
        cout<<" "<<temp[v].fn;
      }

      cout<<endl;
    }
  }

  cout<<endl<<endl;
}
```