```c
#include<stdio.h>
#include<math.h>
#include<stdlib.h>
const double pi=acos(-1.0);
int max(int *h, int rows)
{
    int s=0;
    for(int i=0;i<rows;i++)
    {
        if(s<h[i])
            s=h[i];
    }
    return s;
}
void analytical_method(double *as, double delta_t, int grid_points, int time_step, double k, double interval_length)
{
    for(int i=0;i<grid_points;i++)

        as[i]=sin(2*pi*i*(interval_length/(grid_points-1)))*exp(pi*pi*(-4)*k*(time_step)*delta_t);
}
void print_to_file(double *array,int x_points, double delta_x, char *s)
{
    FILE *fptr=fopen(s,"w");
    fprintf(fptr,"# %s\n", s);
    for(int i=0;i<x_points;i++)
        fprintf(fptr,"%lf \t %lf\n", i*delta_x, array[i]);
    fclose(fptr);
}
void error_method(double *error_array, double *exact_solution, double *num_solution, int grid_points)
{
    for(int i=0;i<grid_points;i++)
        error_array[i]=fabs(exact_solution[i]-num_solution[i]);
}
double error_sum_method(double *error_array, int grid_points)
{
    double error_sum=0;
    for(int i=0;i<grid_points;i++)
    {
        error_sum+=error_array[i];
    }
    return error_sum;
}
void initialization(double *array, int grid_points, double interval_length)
{
    for(int i=0; i<grid_points;i++)
        array[i]=sin(2*pi*i*(interval_length/(grid_points-1)));
}
void numerical_solution(double *u, double *uxx, int grid_points, double delta_t, double interval_length, double k)
{
    //array[i+1][columns-1]=array[i+1][0];
    for(int j=0;j<grid_points;j++)
    {
        u[j]+=k*delta_t*uxx[j];
    }
}
void uxx_calculate(double *uxx, double *u, int grid_points, double interval_length)
{
    uxx[0]=(u[1]-2*u[0]+u[grid_points-2])*(grid_points-1)*(grid_points-1)/(interval_length*interval_length);
    uxx[grid_points-1]=uxx[0];
    for(int i=1; i<grid_points-1;i++)
        uxx[i] = (u[i+1]-2*u[i]+u[i-1])*(grid_points-1)*(grid_points-1)/(interval_length*interval_length);
}
int main(int argc, char *argv[])      //argv[1] = number of grid points, argv[i] = numerical and analytical solution printing time step
{
```

```c
    int grid_points=128;
    int tc=1000;     //time counter parameter. default of 1000
    double delta_t=0.00001;
    double interval_length=1;
    double k=1;
    int print_time_step[20];

    print_time_step[0]=0;
    print_time_step[1]=100;
    print_time_step[2]=500;
    print_time_step[3]=1000;
    int pts_length=4;                //print_time_step array length. Default is 3+1

    if(argc>1)                    //for intializing printing time steps and grid points based on command line inputs
    {
        grid_points=atoi(argv[1]);
        if(argc>2)
        {
            for(int i=2; i<argc;i++)
            {
                print_time_step[i-2]=atoi(argv[i]);
                if(print_time_step[i-2]==0 || print_time_step[i-2]>3000)
                {
                    printf("\n Invalid time steps for printing. Please specify an integer between 1 and 3000 for every printing time step");
                    return -1;
                }
            }
            tc=max(print_time_step, argc-2);
            pts_length=argc-2;
        }
    }



    if(grid_points==0 || grid_points>220)       //the explicit scheme for the given problem with delta_t=0.00001 converges only if the grid points are less than 223 points
    {
        printf("\n Invalid number of grid points. Please specify number of grid points below 1000");
        return -1;
    }

    grid_points+=1;    //to get even number of points since first and last point are one and same

    //double **analytical_solution= (double **)malloc(time_steps * sizeof(double *));
    /*double **num_solution= (double **)malloc(time_steps*sizeof(double *));
    double **error_array= (double **)malloc(time_steps*sizeof(double *));
    for(int i=0;i<time_steps;i++)
    {
        analytical_solution[i]=(double *)malloc(time_steps*sizeof(double));
        num_solution[i]=(double *)malloc(time_steps*sizeof(double));
        error_array[i]=(double *)malloc(time_steps*sizeof(double));
    }*/

    double *u= (double *)malloc (grid_points*sizeof(double));  //solution
    double *uxx= (double *)malloc (grid_points*sizeof(double));  //second derivative with respect to space
    double *as= (double *)malloc (grid_points*sizeof(double));   //analytical solution
    double *error=(double *)malloc(grid_points*sizeof(double));  //for each grid points
    double *error_sum= (double *)malloc((tc+1)*sizeof(double));     //for sum of errors of each grid points at various time steps

    //printf("\n%p",&analytical_solution[0][0]);
    //initialization(analytical_solution[0], grid_points, interval_length);
    //analytical_method(analytical_solution, delta_t, time_steps, grid_points, k);
    //printf("\n%lf", analytical_solution[100][32]);

    initialization(u, grid_points, interval_length);
```

```
    analytical_method(as, delta_t, grid_points, 0, k, interval_length);  //to initialize analytical solution at 0

    //printf("\n%lf",u[33]);
    error_sum[0]=0;
    for(int j=0;j<pts_length;j++)
        if(print_time_step[j]==0)   //to dump the analytical and numerical solution at 0th time step.
            {
                char Numsol[1000], Asol[1000];
                sprintf(Numsol,"Numerical solution at %dth time step for %d grid points.txt", 0, grid_points-1);
                print_to_file(u, grid_points, (double)interval_length/(grid_points-1),Numsol);
                sprintf(Asol,"Analytical solution at %dth time step for %d grid points.txt", 0, grid_points-1);
                print_to_file(as, grid_points, (double)interval_length/(grid_points-1),Asol);
            }

    for(int i=1;i<=tc;i++)
    {
        uxx_calculate(uxx, u, grid_points, interval_length);
        /*if(i==1)
            printf("\n%lf",uxx[32]);*/
        numerical_solution(u, uxx, grid_points, delta_t, interval_length, k); //updates u to the next time step
        /*if(i==100)
            printf("\n%lf",u[32]);*/
        analytical_method(as, delta_t, grid_points, i, k, interval_length);
        /*if(i==100)
            printf("\n%lf",as[32]);*/
        error_method(error, as, u, grid_points);
        /*if(i==1000)
            printf("\n%lf",error[97]);*/
        error_sum[i]= error_sum_method(error, grid_points);
        /*if(i==1000)
            printf("\n%lf",error_sum[grid_points-3]);*/
        for(int j=0;j<pts_length;j++)
            if(print_time_step[j]==i)   //to dump the analytical and numerical solution.
            {
                char Numsol[1000], Asol[1000];
                sprintf(Numsol,"Numerical solution at %dth time step for %d grid points.txt", i, grid_points-1);
                print_to_file(u, grid_points, (double)interval_length/(grid_points-1),Numsol);
                sprintf(Asol,"Analytical solution at %dth time step for %d grid points.txt", i, grid_points-1);
                print_to_file(as, grid_points, (double)interval_length/(grid_points-1),Asol);
            }
    }

    char error_string[]="Sum of errors at each grid point for various time steps.txt";
    print_to_file(error_sum, tc+1, delta_t, error_string);


    free(error_sum);
    free(error);
    free(as);
    free(uxx);
    free(u);
}
```