Laboratory for Parallel Programming
Prof. Dr. Felix Wolf

TECHNISCHE UNIVERSITÄT DARMSTADT

Aamer Shah, shah@cs.tu-darmstadt.de

Large-Scale Parallel Computing (WS 15/16)

Exercise 1

A solution will be presented on October 27th, 2015. Attendance is optional. The exercise solutions will not be corrected and graded.

Task 1

Fibonacci sequence is a popular number series constructed by adding the last two numbers of the sequence to get the next number. The first and the second numbers are fixed as 0 and 1. The series we get is: `0 1 1 2 3 5 8 13 21 ...` Write a C program that prints the Fibonacci series till the n^{th} place. The program should take the input value n on the command line and print an error if it is negative.

a) First, write a program that uses **loops** to generate the sequence.

b) Second, write the same program again such that it uses only **recursion** to generate the sequence.

c) Which implementation of the program is more efficient with respect to execution time and memory.

Solution

Listing 1: A single program with both implementations

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  unsigned long fibonacci_iterative(int n);
5  unsigned long fibonacci_recursive(int n);
6
7  int main(int argc, char* argv[])
8  {
9    if(argc != 2)
10   {
11     printf("Enter the Nth place to display its Fibonacci series value
         \n");
12     return -1;
13   }
14
15   int nth_place = atoi(argv[1]);
16
17   if(nth_place < 0)
18   {
19     printf("Error, the Nth place value should be positive\n");
20     return -1;
21   }
22
23   unsigned long fibonacci_value_i = fibonacci_iterative(nth_place);
24   unsigned long fibonacci_value_r = fibonacci_recursive(nth_place);
25
26   printf("The Nth place Fibonacci series value is iterative: %lu,
       recursive: %lu\n", fibonacci_value_i, fibonacci_value_r);
```

```
27
28    return 0;
29  }
30
31  unsigned long fibonacci_iterative(int n)
32  {
33    unsigned long first = 0;
34    unsigned long second = 1;
35
36    unsigned long nth_value;
37
38    if(n == 0)
39    {
40      return first;
41    }
42
43    if(n == 1)
44    {
45      return second;
46    }
47
48    int i;
49
50    for(i = 2; i <= n; i++)
51    {
52      nth_value = first + second;
53      first = second;
54      second = nth_value;
55    }
56
57    return nth_value;
58  }
59
60  unsigned long fibonacci_recursive(int n)
61  {
62    if(n == 0)
63    {
64      return 0;
65    }
66
67    if(n == 1)
68    {
69      return 1;
70    }
71
72    return (fibonacci_recursive(n - 1) + fibonacci_recursive(n - 2));
73  }
```

The iterative solution is better, both in execution time and memory.

Task 2

Write a C program that reads a file containing two matrices and writes their sum back to a new file. The input file should be named `input.dat`. The first line of the file should describe the number of rows and columns of the matrices. The rest of the file should contain the two matrics, one after the other. The output file should be named `output.dat` and should have the same format, i.e; the first line describing

the dimensions, followed by the sum of the matrices. The program should internally store the matrices as two dimensional arrays. The matrices can be assumed to be of integer data type.

a) Implement the program, and use a function that takes the two matrices as input and calculates their sum.

b) Re-implement the function such that it still takes the two matrices as input, but does not use pointers for passing the input matrices (Don't use structs or any other derived data types). **Hint:** There is a difference between int `**array`, `int *array[N]` and `int array[N][M]`.

c) Is a solution to part (b) possible in C++?

Solution

Listing 2: A single program with both implementations

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void mat_sum_p(int rows, int cols, int** mat_1, int** mat_2, int**
       sum);
5  void mat_sum_s(int rows, int cols, int mat_1[rows][cols], int mat_2[
       rows][cols], int sum[rows][cols]);
6  void print_mat(int rows, int cols, int** mat);
7  void print_mat_s(int rows, int cols, int mat[rows][cols]);
8  void write_mat(FILE* fh, int rows, int cols, int** mat);
9  void write_mat_s(FILE* fh, int rows, int cols, int mat[rows][cols]);
10
11 int main(int argc, char* argv[])
12 {
13   int rows, cols;
14   FILE* fh = fopen("input.dat", "r");
15
16   fscanf(fh, "%d%d", &rows, &cols);
17
18   printf("Matrix size: %d x %d\n", rows, cols);
19
20   int r, c;
21   int** mat_1, **mat_2, **sum;
22
23   int mat_1s[rows][cols], mat_2s[rows][cols], sum_s[rows][cols];
24
25   mat_1 = malloc(sizeof(int*) * rows);
26   mat_2 = malloc(sizeof(int*) * rows);
27
28   sum = malloc(sizeof(int*) * rows);
29
30   for(r = 0; r < rows; r++)
31   {
32     mat_1[r] = malloc(sizeof(int) * cols);
33     sum[r] = malloc(sizeof(int) * cols);
34
35     for(c = 0; c < cols; c++)
36     {
37       fscanf(fh, "%d", &mat_1[r][c]);
38       mat_1s[r][c] = mat_1[r][c];
39     }
```

```
40    }
41
42    for(r = 0; r < rows; r++)
43    {
44      mat_2[r] = malloc(sizeof(int) * cols);
45
46      for(c = 0; c < cols; c++)
47      {
48        fscanf(fh, "%d", &mat_2[r][c]);
49        mat_2s[r][c] = mat_2[r][c];
50      }
51    }
52
53    fclose(fh);
54
55    mat_sum_p(rows, cols, mat_1, mat_2, sum);
56    mat_sum_s(rows, cols, mat_1s, mat_2s, sum_s);
57
58    printf("Matrix 1:\n");
59    print_mat(rows, cols, mat_1);
60
61    printf("Matrix 2:\n");
62    print_mat(rows, cols, mat_2);
63
64    printf("Matrix sum:\n");
65    print_mat(rows, cols, sum);
66
67    printf("Matrix sum (array):\n");
68    print_mat_s(rows, cols, sum_s);
69
70    fh = fopen("output.dat", "w");
71
72    write_mat(fh, rows, cols, sum);
73
74    fclose(fh);
75
76    fh = fopen("output_s.dat", "w");
77    write_mat_s(fh, rows, cols, sum_s);
78    fclose(fh);
79
80    return 0;
81  }
82
83  void mat_sum_p(int rows, int cols, int** mat_1, int** mat_2, int**
      sum)
84  {
85    int r, c;
86
87    for(r = 0; r < rows; r++)
88    {
89      for(c = 0; c < cols; c++)
90      {
91        sum[r][c] = mat_1[r][c] + mat_2[r][c];
92      }
93    }
94  }
```

```
95  void mat_sum_s(int rows, int cols, int mat_1[rows][cols], int mat_2[
        rows][cols], int sum[rows][cols])
96  {
97    int r, c;
98
99    for(r = 0; r < rows; r++)
100   {
101     for(c = 0; c < cols; c++)
102     {
103       sum[r][c] = mat_1[r][c] + mat_2[r][c];
104     }
105   }
106 }
107
108 void print_mat(int rows, int cols, int** mat)
109 {
110   int r, c;
111
112   for(r = 0; r < rows; r++)
113   {
114     for(c = 0; c < cols; c++)
115     {
116       printf("%d\t", mat[r][c]);
117     }
118     printf("\n");
119   }
120 }
121
122 void print_mat_s(int rows, int cols, int mat[rows][cols])
123 {
124   int r, c;
125
126   for(r = 0; r < rows; r++)
127   {
128     for(c = 0; c < cols; c++)
129     {
130       printf("%d\t", mat[r][c]);
131     }
132     printf("\n");
133   }
134 }
135
136 void write_mat_s(FILE* fh, int rows, int cols, int mat[rows][cols])
137 {
138   int r, c;
139   for(r = 0; r < rows; r++)
140   {
141     for(c = 0; c < cols; c++)
142     {
143       fprintf(fh, "%d ", mat[r][c]);
144     }
145     fprintf(fh, "\n");
146   }
147 }
148
149
```

```
150 void write_mat(FILE* fh, int rows, int cols, int** mat)
151 {
152   int r, c;
153   for(r = 0; r < rows; r++)
154   {
155     for(c = 0; c < cols; c++)
156     {
157       fprintf(fh, "%d ", mat[r][c]);
158     }
159     fprintf(fh, "\n");
160   }
161 }
```

It is not possible in C++, as only C allows to use automatic variables as array size.