AC 2012-3155: USING MODULAR PROGRAMMING STRATEGY TO PRACTICE COMPUTER PROGRAMMING: A CASE STUDY

Dr. Wangping Sun, Oregon Institute of Technology
Dr. Xin Wang, Oregon Insitute of Technology
Miss Xian Sun

©American Society for Engineering Education, 2012
Using Modular Programming Strategy to Practice Computer Programming: a Case Study

Abstract

It is important for the engineering and technology students to learn and improve their computer programming skills throughout their college education. To make the students’ learning process more efficient, a Modular Programming Strategy (MPS) was introduced at the last ASEE annual conference in 2011. As an implementation of the MPS, the authors of this paper will present a case study to demonstrate how this programming concept is applied in the programming assignments of a graduate course. This graduate course was offered to ten graduate students in the area of manufacturing engineering technology. All of them were engineering students without computer science, electrical engineering or software engineering background. Through this case study, the authors will show that the MPS will enable the students to quickly solve many engineering problems in a correct and timely manner.

1. Introduction

Computer programming is an essential and integral part of any engineering program\(^1\), \(^2\). It is important for the engineering and technology students to learn and improve their computer programming skills throughout their college education\(^3\). Computer programming involves substantial hands-on practice. Problem-based learning is an effective way to improve computer programming skills, in which the students are guided through a set of steps allowing them to explore and understand the problem, generate potential solutions, investigate them and come up with the most effective solution\(^4\).

At our university, graduate engineering students face the task of solving problems by using numerical approaches in their theses and in more than half of their courses. Proficient programming skills will enable them to solve these problems correctly and timely. To make the students’ learning process more efficient, a pedagogic model based on modular programming concept was introduced at the last ASEE annual conference in 2011\(^5\). In that paper, the authors stated that with the Modular Programming Strategy (MPS), the students will be able to develop computer code more easily. As an implementation of the MPS, this paper will present a case study to demonstrate how this programming concept is applied in the programming assignments of a graduate course.

2. A brief review on the Modular Programming Strategy (MPS)

In the engineering world, a complex project is always divided into small pieces. To finish the whole project, individual engineers must work on each piece and the team lead must be
responsible for integrating every piece together\(^5\). The same strategy has been successfully used in software industry and is called “modular programming”, by which a complicated programming assignment is divided into small, manageable procedures\(^6\). To use the MPS effectively, three requirements have to be ensured\(^7\). Firstly, a main procedure always needs to be created to coordinate and integrate the work of all individual procedures. Secondly, all the sub-procedures only communicate with their direct superior procedures. Thirdly, any “cross-talk” among the sub-procedures at the same level is not allowed. Figure 1 shows two examples of the MPS, in which diagram a) on the left shows a correct MPS model and diagram b) on the right shows an incorrect one. The correct communication is represented by solid arrowed lines and the prohibited communication is represented by dashed arrowed lines.

![Diagram of correct and incorrect MPS models](image)

**Figure 1.** Examples of the MPS.

The MPS breaks a complex problem into small pieces and provides a roadmap to solve that problem in a straightforward way. By using the MPS, a clear and concise programming structure can be easily built\(^5\). Moreover, the MPS is not confined by any specific programming language. It can be used by Visual Basic, MATLAB or other languages. Here is a programming example using VBA (Visual Basic Applications) in the sophomore class *ENGR 266 Computer Programming for Engineers*:

\[
\sin(x) + \cos(x) = \sum_{i=0}^{n} \frac{(-1)^i x^{2i+1}}{(2i + 1)!} + \sum_{i=0}^{n} \frac{(-1)^i x^{2i}}{(2i)!}
\]

To program the above question, the first step is to set up a commander (i.e. the main procedure) that coordinates and integrates all the intermediate results and generates the final solution. After the main procedure is designed, the detail programming can be divided into 2 parts, which includes the computation of the factorial and the summation of the sequences. The modular structure of the program is shown in Figure 2. In this figure, the arrowed lines represent the communication between a sub-procedure and its superior. Please note that the MPS does not...
allow the communications between sub-procedures at the same level. If the communication is needed between them, it will be coordinated by the sub-procedures’ common superior.

Figure 2. Modular structure of a VBA programming example.

After the structure is developed, the programming process will become very simple. The whole program is illustrated in Figure 3.

![Diagram of modular structure](image)

**Figure 3.** VBA code based on the modular structure in Figure 2.

```vba
Public Sub main()
    Dim n As Integer 'accuracy of the number
    Dim x, sin_value, cos_value, a As Double
    x = 3.1415
    n = 10
    sin_value = Sine_Computation(x, n) 'compute sin
    cos_value = Cosine_Computation(x, n) 'compute cosine
    a = sin_value + cos_value
End Sub

Public Function Sine_Computation(x, n) 'compute sine
    Dim i As Integer 'counter
    For i = 0 To n
        a = Factorial(i + i + 1)
        sum = sum + (-1) ^ i * (x ^ (2 * i + 1)) / a
    Next
    Sine_Computation = sum 'return to commander
End Function

Public Function Cosine_Computation(x, n) 'compute cosine
    Dim i As Integer 'counter
    For i = 0 To n
        a = Factorial(i + i)
        sum = sum + (-1) ^ i * (x ^ (2 * i + 1)) / a
    Next
    Cosine_Computation = sum 'return to commander
End Function

Public Function Factorial(a) 'calculate factorial
    Dim i As Integer 'counter
    Dim f As Double 'hold factorial number
    i = 1 'initial value of i
    If a = 0 Then 'factorial 0 = 1
        Factorial = 1 'return factorial to superior
    Else 'compute factorials of non-zero numbers
        For i = 1 To a
            i = i + 1
        Next
        Factorial = f 'return factorial to superior
End Function
```
3. A case study to implement the MPS

The MPS has not only been used in the programming course at the sophomore level, but also been implemented in a graduate level course, MFG 596 Plant Design and Materials Handling Systems. This course introduces the design of industrial facilities with the emphasis on manufacturing engineering and materials handling (including systematic layout planning, activity relationship chart, materials handling principles and equipment, unit load concept, flexible manufacturing system, warehouse components, etc.).

In this course, students are required to use MATLAB to implement five optimization algorithms, one of which is Direct Cluster Algorithm (DCA). This algorithm helps form product families, which is the first step to form work cells in agile manufacturing (please note that this algorithm cannot be simulated or solved by the discrete event simulation software, such as Arena®). The DCA forms product families by sequentially moving the rows upward and the columns leftward in the machine-product matrix. Its implementation steps are as follows:

- Order the rows in descending order
- Order the columns in ascending order
- Sort the columns: start from the first row; iterate through the first, second, third … column in the row; if there is “1” met, move (sort out) the whole column leftward and right next to the last column just sorted out; if there is nothing in the intersection of the row and column, no movement is made; repeat the iteration until the all the columns are sorted out.
- Sort the row: start with the first column; iterate through the first, second, third … row in the column; if there is “1” met, move (sort out) the row upward and right below the last row just sorted out; if there is nothing in the intersection of the row and column, no movement is made; repeat the iteration until all the rows are sorted out.
- Form work cells by following the principle that machines in each cell are dedicated to produce a complete component.

The following example explains how the DCA is implemented: a manufactured product contains 6 parts and needs 5 machines to process. The information is shown in Figure 4. For example, part 1 needs machines 1 and 3. Machine 5 processes part 3 and part 6. Part 2 does not need any machine. The question is how the work cells can be formed.

<table>
<thead>
<tr>
<th>part</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>1</td>
<td></td>
<td>1</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>sum</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>
The formation of work cells starts from summing up each row and each column, as shown in the last row and last column in Figure 4.

The second step is to order the rows in descending order based on the sum values of the rows, as shown in Figure 5. Note that part 3 goes up to the first row (above all other parts) in the table.

<table>
<thead>
<tr>
<th>part</th>
<th>machine 1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>sum</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 5. Column and row ordering.

The third step is to order the columns in ascending order based on sum values of the columns. In this example, since all the sum values of the columns are the same, which is two, the order of the columns does not change, as shown in Figure 5.

The fourth step is to sort the columns. It starts from the first row where part 3 is located, as shown in Figure 5. Every intersection of this row with the columns will be examined. First, the intersection with the first column (where machine 1 is located) is examined. There is nothing there. So, no movement is made. Next, the intersection with the second column (where machine 2 is located) is examined. There is a “1”. So, move (sort out) this column to the left, as shown in Figure 6.

<table>
<thead>
<tr>
<th>part</th>
<th>machine 2</th>
<th>1</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>sum</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 6. Column sorting.

After that, the intersection with the third column (where machine 3 is located) is examined. There is nothing there, so no movement is made. Following that, the intersection with the fourth column (where machine 4 is located) is examined. There is a “1” there. So, move (sort out) this column to the left, as shown in Figure 7.
Finally, the intersection with the fifth column (where machine 5 is located) is examined. There is a “1” there, so move (sort out) this column to the left. By now, the iteration ends and three columns (where machines 2, 4 and 5 are located) are sorted out as shown in Figure 8. Repeat the same iterations on the fourth and the fifth column, where machines 1 and 3 are located. The fourth column (where machine 1 is located) is sorted out first. The fifth column (where machine 3 is located) is sorted out lastly. The final result of column sorting is shown in Figure 8.

The fifth step of the DCA is to sort the rows. It starts from the first column in Figure 8 (where machine 2 is located). Every intersection of this column with the rows will be examined. First, the intersection with the first row (where part 3 is located) is examined. There is a “1” there, so move (sort out) this row upward. Since part 3 is already in the first row, no movement is made. Next, the intersection with the second row (where part 5 is located) is examined. There is a “1” there, so move the row upward. Since part 5 already is in the second row and the first row is already sorted out, no movement is made. Since the last three intersections in the first column (where machine 2 is located) have nothing there, the iteration ends. By now, two rows (where parts 3 and 5 are located) are sorted out. Repeat the same iterations on the third to the sixth row (where the parts 6, 1, 4, and 2 are located) until all the rows are sorted out. The result of row sorting is shown in Figure 9. Even though Figure 8 and Figure 9 look the same, Figure 9 is the result after all the rows have been sorted out.

The final result after sorting is also shown in Figure 9. It can be seen that two work cells can be formed. Cell one: machines 2, 4, and 5 for parts 3, 5 and 6. Cell two: machines 1 and 3 for parts
1 and 4. Part 2 can be attached to either of these two cells. In this example, it is attached to Cell two.

<table>
<thead>
<tr>
<th>machine</th>
<th>2</th>
<th>4</th>
<th>5</th>
<th>1</th>
<th>3</th>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>part</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>sum</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

**Figure 9.** Formation of work cells.

With the help of the MPS, a neat and simple program structure can be developed. As shown in Figure 10, one main procedure and five sub-procedures are designed for this algorithm. Based on the structure in Figure 10, MATLAB programs can be developed as shown in Figure 11.

**Figure 10.** The programming structure to implement the DCA.
Figure 11. MATLAB programs to implement the MPS.

```matlab
% add machine and part numbers
function [M0]=add_m0_part_numbers(M)
    [m,n]=size(M);
    i=[0:n];
    j=1:m;
    M0=[i';j';M];

% this program implement DCA
clear all
M=load('data.txt') & load original matrix
M0=add_m0_part_numbers(M)
M1=order_rows(M, M0);
[m,n]=size(M1);
M2=order_columns(M1, M0);
M3=order_columns(M2, M0)
M12=sort_rows(M1, M0)

% sort rows
function [M12]=sort_rows(m, n, M10)
    M11=M10;
    B2=[];
    N2=1;
    for j=2:m:
        for i=2:n:
            if M11(i,j)==1:
                B2=[B2; M11(i,:)];
                M11(i,2:n)=2;
                end
        end
        M12=[M11(1:m,1) N2];

% order columns in de
function[M10]=order_columns(m, n, M0)
    M10=M0;
    B1=[];
    N1=0;
    for i=1:n:
        if M0(:,i)==1:
            B1=[B1, M0(:,1)];
            N1=N1+1;
        end
    end
    M10=[M10(:,1) N1];

% order columns in de
function[M10]=order_columns(m, n, M0)
    M10=M0;
    B1=[];
    N1=0;
    for i=1:n:
        if M0(:,i)==1:
            B1=[B1, M0(:,1)];
            N1=N1+1;
        end
    end
    M10=[M10(:,1) N1];
```

Figure 11. MATLAB programs to implement the MPS.
To test the code, a test case is generated as shown in Figure 12. After running the program, it can be found that three work cells can be formed (machines 1, 4 and 6 to produce parts 1, 7 and 2; machines 2 and 7 to produce parts 3 and 4; machines 3 and 5 to produce parts 5 and 6).

![Matrix](image)

To test the code, a test case is generated as shown in Figure 12. After running the program, it can be found that three work cells can be formed (machines 1, 4 and 6 to produce parts 1, 7 and 2; machines 2 and 7 to produce parts 3 and 4; machines 3 and 5 to produce parts 5 and 6).

During one academic quarter, the students were required to program five algorithms at the similar difficulty level. All the students finished their assignments satisfactorily and their frustration from these assignments was kept under a controllable level. The programming proficiency of the students also improved significantly by the end of the term.

4. Conclusions

Computer programming is an intensive, hands-on design process. Modular Program Strategy (MPS) makes the computer programming process less challenging. In this paper, the authors have provided a case study to show how an algorithm in manufacturing area can be converted into computer code in a correct and timely manner. This paper has also provided another supporting case to illustrate that the MPS is versatile and can be used in many areas. Currently, the authors are working with Klamath Union High School, trying to implement the MPS concept in a high school senior project.
References


