IMPLEMENTATION METHOD "DIVIDE AND IMPERA" USING OBJECT -ORIENTED PROGRAMMING IN C

PhD. Student engineer Cătălin LUPU
Ministry of Administration and Interior, Suceava, Romania
lupucata@yahoo.com
Assoc. Prof. PhD.Valeriu LUPU
„Stefan cel Mare" University, Suceava, Romania
valeriul@seap.usv.ro
Professor of Informatics Petru ARCAN
National College of Commerce , Chisinau, Moldova
arptr_us@yahoo.com

Abstract:
In this article presents applications of "Divide et impera" method using object -oriented programming in C #. Main advantage of using the "divide et impera" cost in that it allows software to reduce the complexity of the problem, sub-problems that were being decomposed and simpler data sharing in smaller groups of data (eg sub -algorithm QuickSort). Object-oriented programming means programs with new types that integrates both data and methods associated with the creation, processing and destruction of such data. To gain advantages through abstraction programming (the program is no longer a succession of processing, but a set of objects to life, have different properties, are capable of specific actions and interact in the program). Spoke on instantiation new techniques, derivation and polimorfismul object types.

Keywords: Divide et impera, constructor, destructor, class, member, court

JEL Classification: C61

1. INTRODUCTION TO THE FIELD - THE PRESENTATION METHOD "DIVIDE AND IMPERA"

The method of programming Divide and IMPERA problem lies in dividing the initial size [n] into two or more smaller. In general, is running sub -problems dividing into two approximately equal size, namely [n / 2]. Are Sub -problems sharing in place until their size is sufficiently small to be resolved either directly (if basic). After solving the two runs sub -problems phase of combining the results to solve the whole problem.

Divide and IMPERA method can be applied to solve a problem that meets the following conditions:

- can decompose in (two or more) sub -problems;
- these sub-problems are an independent against another (no sub -problem a resolve on the other and does not use the other);
- these sub-problems these are similar to the initial problem;
- sub-problems in turn can decompose (if necessary) in other sub -problem simpler;
- these simple sub-problem can immediately solve the simplified algorithm.

Because few problems meeting the conditions above, the application of the method is quite rare.

As the name suggests "separating and rule" steps to resolve a problem (called the initial problem) in Divide and IMPERA are:

- decomposition problem initial sub -problems independent, similar to the basic problem, smaller;
- gradual decomposition of sub-problems other sub-problem increasingly simple, until you can resolve this through simplified algorithm;
- solving sub-problems simple;
- combination of solutions found for building solutions sub -problems sizes increasingly large;

- combining the latest solutions getting determine the¬ initial problem solution. Method Divide and IMPERA recurring implementation, because sub-problems initial problem are similar, but smaller.

The basic principle of recursivity is a recall sub-programs when it is active, what happens at a level happens at every level, taking care to ensure the condition of termination of repeated calls. Similar happens in the case method divide and Imperia, to a certain level are two possibilities:

- it¬ was a (sub) problem which allow a simple solution immediately, in which case is resolved (sub) problem and return the call (to subproblema earlier, the larger);
- it was a (sub) which does not allow for an immediate solution, in which ¬ case the decomposition in two or more subprobleme and each of them continues recursive calls (of the procedure or function).

In the final stage of the method and divide Imperia is produced combining subproblemelor (already resolved) by sequences of return calls from recurring. Stages method divide and Imperia (shown above) can be represented by the following general subprogramme (procedure or function) recurring expressed in natural language:

Subprogramme divimp (probably);

If the problem is simple prob
Then resolve to obtain and soil solution
Otherwise i = 1, k running DIVIMP (probably) and get ground 1;
Combine ground solutions 1, ... , Soil K and gain ground;
end_sub-program;

So subprogram divimp is calling for the initial test session, to allow the decomposition k subprobleme simple, for they redials recurring subprogram; finally combine these solutions k subprobleme.

Usually, the initial problem fester in the second subprobleme simple, in this case stages general method of divide and Imperia may be, the pseudo-code language, by a recursive procedure as follows:

```
The procedure divimp (li, ls, soil);
   If ((ls-li) <= eps)
   Then REZOLVATE (li, ls, soil);
   else
     divide (li, m, ls);
     divimp (li, msol1);
     divimp (m, ls, sol2);
     combine (sol1, sol2, soil);
End_procedure;
```

Divimp procedure is called for the initial size of which is the lower limit (li) and the lower limit (ls), if (as) the problem is simple (ls-li <= eps), then the procedure is the solution solves them immediately and return occurs recursive call, if (as) the problem is (still) complex, then divide a procedure divided into two sub-problems, choosing between the position I have and ls, for each of the two sub-problems it redials recurring procedure divimp; finally returns Call of producing a combination of the two solutions sol1 and sol2 by combining the procedure call.

TOWERS OF HANOI PROBLEM

PRESENTATION SOLVING ALGORITHM

Whether three vertical sticks marked A, B, C. On A rod can be found seated in different diameter discs, in ascending order of Diameter on the top down. Initially, rods B and C are empty. To show all the movements that discs on the rod to move the rod B, in the same order, using as a rod for maneuver C and within the following rules:
- Every step to move a single disc;
- A disc can only settle over a disc with a diameter greater.

Resolving this problem is based on the following considerations logical:
- If n = 1, then move it immediately A->B (move on the disc A B);
- Where n = 2, then the string of moves is: A->C, A->B, C->B;
- If n> 2 proceed as follows:
- Dumb (n-1) A->C discs;
- Move a disc A->B;
- Move the (n-1) discs C->B.

We see that the initial problem fester into three sub-problems simpler similar problem of a dumb (n-1) A->C discs, the latest move disc B, move the (n-1) discs C -> B. Sub-problems these dimensions are: n-1, 1, n-1.

These sub-problems are independent, as originally rods (which are disks), rods and final intermediate rods are different. Note H (n, A, B, C) = string of move s discs from an A to B, using C.

for
  n = 1, A->B
  n> 1, h (n, a, b, c) = h (n-1, a, c, b), ab, h (n-1, c, b, a)
end_for

Visual Basic version

```
Sub Button20_Click()
Dim n As Integer, a As sir, b As sir, c As sir
d = ""
a.s = "A"
b.s = "B"
c.s = "C"
n = InputBox("n=", ib_title)
hanoi n, a, b, c
MsgBox d
End Sub
Sub hanoi(n As Integer, a As sir, b As sir, c
As sir)
If n = 1 Then
d = d + "(" + a.s + "->" + b.s + "),"
Else
```

C# version

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication1
{
   class var_globale
   {
      public int n;
      public string sir;
      public int[] vector;

   }
   class Program
```

```
hanoi n - 1, a, c, b
d = d + "(" + a.s + "->" + b.s + "),"
hanoi n - 1, c, b, a
End If
End Sub
```

```
    {


        static void cit_n(string mes, ref int
x)
        {
            do
            {
                Console.WriteLine(mes);
          x = int.Parse(Console.ReadLine());
            } while (x < 0 || x > 100);
        }


        static void hanoi(int n, char a, char
b, char c)
        {
            if (n == 1)
                Console.WriteLine(" {0} {1}
", a, b);
            else
            {
                hanoi(n - 1, a, c, b);
                Console.WriteLine(" {0} {1}
", a, b);
                hanoi(n - 1, c, b, a);
            }
        }




     static void Main(string[] args)
        {
var_globale v_g = new var_globale();
cit_n("N = ", ref v_g.n);
char a, b, c;
a = 'a';
b = 'b';
```

```
c = 'c';
hanoi(v_g.n, a, b, c);
Console.ReadLine();
      }
   }
}
```

Sort QUICK (QUICKSORT)

In a painting is completed by elements in real numbers. To ascending order using the method of sorting fast. The problem is based on the following steps implemented in the pr imary:

- procedure is called "quick" with the lower limit were = 1 and the ¬ upper limit ls = n;
- the "poz" done in moving the item [s] on exactly what ¬ position it will occupy in the final vector ordered, the "poz" return (in k) position of this item;
- in this way, vector V is divided into two parts: li ¬ ... k-1 and k 1 ... ls;
- for each of these parties redials the procedure is ¬ "quick", with the modified accordingly;
- in this way, the first element in ¬ each part will be positioned exactly the position that a final deal will finally ordered the vector (the "poz");
- each of the two parties will thus divided ¬ into two parts, the process continues until the parties come to overlap, indicating that all elements of the vector have been moved on exactly what positions they occupy in the final vector; vector is so ordered;
- at this time to produce your recursive calls and the implementation of its ends.

Remarks:

- If the item is left, then compared with elements of the right and jump (j: = j -1) elements of greater than him;
- If the item is right, then compared with elements of the left and jump (i: = i +1) elements smaller than him.

Visual Basic version

```
Sub Button18_Click()
Dim n As Integer, a As vector
cit_n "n = ", n
cit_date "a", n, a
'MsgBox "Sirul a  este"
tipar "Sirul a  este", n, a
divimp 1, n, a
'MsgBox "Sirul a sortat este"
tipar "Sirul a sortat este", n, a
End Sub

Sub cit_n(mes As String, n As Integer)
Do
n = InputBox(mes, y)
```

C# version

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication1
{
   public interface sortt
   {
      int n
      {
         get;
         set;
      }
```

```
Loop Until n > 0 And n < 100
End Sub

Sub cit_date(mes As String, n As Integer, a As vector)
For i = 1 To n
a.v(i) = InputBox(mes + "(" + Str$(i) + ")=", y)
Next
End Sub

Sub tipar(mes As String, n As Integer, a As vector)
sir = ""
For i = 1 To n
sir = sir + Str$(a.v(i)) + ","
Next
MsgBox mes + " " + sir
End Sub
Sub sort(p As Integer, q As Integer, a As vector)
Dim m As Integer
If a.v(p) > a.v(q) Then
m = a.v(p)
a.v(p) = a.v(q)
a.v(q) = m
End If
End Sub

Sub divimp(p As Integer, q As Integer, a As vector)
Dim m As Integer
If (q - p) <= 1 Then
sort p, q, a
Else
m = Int((p + q) / 2)
divimp p, m, a
divimp m + 1, q, a
interc p, q, m, a
End If
End Sub

Sub divimp(p As Integer, q As Integer, a As vector)
```

```
    int[] a
    {
        get;
        set;
    }
    string sir
    {
        get;
        set;
    }
}
public class sortta : sortt
{
    private int _n;
    private int[] _a;
    private string _sir;

    public void cit_n()
    {
        do
        {
            Console.WriteLine("N = ");
            _n = int.Parse(Console.ReadLine());
        } while (_n < 0 || _n > 100);
    }
    public void cit_date()
    {

        _a = new int[_n+1];
        for (int i = 1; i <= _n; i++)
        {
            Console.WriteLine("a[{0}]= ", i);
            _a[i] = int.Parse(Console.ReadLine());
        }
    }
    public  void tip_date()
    {
        Console.WriteLine(_sir);
        // aa = new int[n];
```

```
Dim m As Integer
If (q - p) <= 1 Then
sort p, q, a
Else
m = Int((p + q) / 2)
divimp p, m, a
divimp m + 1, q, a
interc p, q, m, a
End If
End Sub
```

```
        for (int i = 1; i <= _n; i++)
        {
            Console.WriteLine("a[{0}]= {1}, ", i,
_a[i]);
            // aa[i] =
int.Parse(Console.ReadLine());
        }

    }
    public void sortare()
    {
        int i, k, x;
        do
        {
            k = 0;
            for (i = 1; i <= _n - 1; i++)
                if (_a[i] > _a[i + 1])
                {
                    x = _a[i];
                    _a[i] = _a[i + 1];
                    _a[i + 1] = x;
                    k = 1;
                }
        } while (k == 1);
    }
    public void insertie_directa()
    {
        int i,j,r;
        for(i=2;i<=_n;i++)
        {
            _a[0]=_a[i];
            j=i-1;
            r = 1;

            while ((_a[j]>_a[0])&&(r==1))
            {
                _a[j+1]=_a[j];
                j=j-1;
                if (j <= 0)
                {
                    r = 0;
                                }
```

```
        }
        _a[j+1]=_a[0];
    }
}
public void insertie_binara()
{
    int i, j, s, d, m, x;
   // _a = new int[100];
    for (i = 2; i <= _n; i++)
    {
        x = _a[i];
        s = 1;
        d = i - 1;
        while (s <= d)
        {
            m = (int)((s + d) / 2);
            if (_a[m] > x)
                d = m - 1;
            else
                s = m + 1;
        }
        for (j = i - 1; j >= s; j--)
            a[j + 1] = a[j];
        a[s] = x;
    }
}
public void selectie()
{
    int i, j, k,x;
    for (i = 1; i <= n - 1; i++)
    {
        k = i;
        x = _a[i];
        for (j = i + 1; j <= _n; j++)
            if (_a[j] < x)
            {
                x = _a[j];
                k = j;
            }
        _a[k] = _a[i];
        _a[i] = x;
```

```
    }
}
public void selectie_performanta()
{
   int i, j,min, x;
   for (i = 1; i <= n - 1; i++)
   {
      min = i;
      for (j = i + 1; j <= _n; j++)
         if (_a[j] < _a[min])
         {
            min = j;
         }
      x=_a[min];
      _a[min] = _a[i];
      _a[i] = x;
   }
}

public void sortarea_prin_amestecare()
{
   int j,k,l,r, x;
   l = 2;
   r = _n;
   k = _n;
   do
   {

   for (j = r; j>=1; j--)
      if (_a[j-1] > _a[j])
         {
      x = _a[j-1];
      _a[j-1] = _a[j];
      _a[j] = x;
         k=j;
         }
      l=k+1;
      for(j=1;j<=r;j++)
         if (_a[j-1] > _a[j])
         {
      x = _a[j-1];
```

```
      _a[j-1] = _a[j];
      _a[j] = x;
         k=j;
          }
      r=k-1;
   } while(!(l>r));
}
public void shellsort()
{
   int i, j, k, s, x, b;
   k = _n;
   do
   {
      k = k / 2;
      do
      {
         b = 1;
         for (i = 1; i <= _n - k; i++)
            if (_a[i] > _a[i + k])
            {
               x = _a[i];
               _a[i] = _a[i + k];
               _a[i + k] = x;
               b = 0;
            }
      } while (b == 0);
   } while (k != 1);
}
public void quicksort_recursiv()
{
   int m;
   m = 1;
   sortare(m, _n);
}
public void sortare(int s, int d)
{
   int i, j, x, w;
   i = s;
   j = d;
   x = _a[(s + d) / 2];
   do
```

```
    {
        while (_a[i] < x)
            i = i + 1;
        while (_a[j] > x)
            j = j - 1;

        if (i <= j)
        {
            w = _a[i];
            _a[i] = _a[j];
            _a[j] = w;
            i = i + 1;
            j = j - 1;
        }
    } while ((i <= j));
    if ( s<j)
        sortare(s, j);
    if (i < d)
        sortare(i, d);

}
public int n
{
    get
    {
        return _n;
    }
    set
    {
        _n = value;
    }
}
public string sir
{
    get
    {
        return _sir;
    }
    set
    {
        _sir = value;
```

```
            }
        }
        public int[] a
        {
            get
            {
                return _a;
            }
            set
            {
                _a = value;
            }
        }

    }
    class Program
    {

        static void Main(string[] args)
        {
            sortta SORTTA = new sortta();
            SORTTA.cit_n();
            Console.WriteLine("N = {0}",
SORTTA.n);
            SORTTA.a = new int[100];
            SORTTA.sir = "Vectorul initial este : ";
            SORTTA.cit_date();
            SORTTA.tip_date();
//            SORTTA.sortare();
            //SORTTA.insertie_directa();
            //SORTTA.insertie_binara();
            //SORTTA.selectie();
//            SORTTA.selectie_performanta();
            //SORTTA.sortarea_prin_amestecare();
            //SORTTA.shellsort();
            //SORTTA.heapsort();
            SORTTA.quicksort_recursiv();
            SORTTA.sir="Vectorul sortat este : ";
            SORTTA.tip_date();
            Console.ReadLine();
        }
```

```
    }
}
```

CONCLUSIONS

This article presents the advantages of "divide et impera" method and object-oriented programming. One of the major advantages of object oriented programming is the ability to reuse existing code. Can design new classes using already constructed class is called inheritance. If a class A inherits class B, then B and class methods will be considered as belonging to class A. Heritage and create a class "base" in order to store the common characteristics of different classes, such properties will not be specified in each class separately. Main advantages of using object oriented programming are:

- ease of design and code reuse:

Once tested the correctness of the operation objects of an application, they can be used without any problem in another application. This advantage can be exploited by the formation of libraries of objects. Regarding design, it facilitates the decomposition of complex problems in simple sub-problems, which can be easily modeled using objects (variables will describe properties of objects shaped their actions and methods).

- abstraction:

Designers can obtain an overall view to the behavior of objects and interactions between them, the details are buried in the composition of objects.

- safety data:

Able objects to behave like "black boxes", it can be used without the knowledge of their composition, providing privacy and decreases the frequency used appearances and the errors related to wrong handling of types of data.

REFERENCES

1. Dorel Lucanu - *Bazele proiectării programelor şi algoritmilor II: Tehnici de programare* - Editura Universităţii "Al. I. Cuza" , Iaşi, 1996
2. Cristian Masalagiu, Ioan Maxim, Ioan Asiminoaei - *Metodica predării informaticii* - Editura Matrix Rom, Bucureşti, 2001
3. Tudor Sorin - *Tehnici de programare* - Editura L&S Infomat, Bucureşti, 1994
4. Tudor Sorin - *Tehnici de programare* - Editura Teora, Bucureşti, 1994
5. Lucian Sasu - *Visual C#,2005*
6. Valentin Cristea - *Tehnici de programare* - Editura Teora, Bucureşti, 1993
7. Ioan Odăgescu - *Metode şi tehnici de programare* - Editura Intact, Bucureşti, 1994
8. Ioan Odăgescu - *Olimpiadele naţionale şi internaţionale de informatică* - Editura L&S Infomat, Bucureşti, 1996
9. Mihai Oltean - *Culegere de probleme* - Editura Libris, Cluj, 1997
10. Mihai Mocanu - *333 probleme de programare* - Editura Teora, Bucureşti, 1993
11. Diana Gruiţă - *Culegere de probleme* - Editura Libris Agora, Cluj, 1998
12. **** Programarea Orientata pe Obiecte si Programarea Vizuala cu C# .Net – documentatie