Design and Implementation of an Efficient Algorithm Using Data Structures: A Recipe for the Structured Process Called Top Down Programming

Chukwudi Igbe

Elei Florence .O

Department of Computer Science,
Imo State University Owerri, Nigeria

Doi:10.5901/jesr.2013.v3n9p17

Abstract

Top-down programming is procedural programming style, which design begins by defining the solution at the highest level of functionality and breaking it down further and further into small routines that can be easily documented and coded . The main aim of this paper is a revisiting of the generic top down programming approach in solving and implementation of an efficient algorithm. we went into these research because programming is traditionally taught using bottom up approach where details of syntax and implementation of data structure are the predominant concept. The top down approach proposed focuses instead on understanding the abstractions represented by the classical data structures without regard to their physical implementation. This paper discusses the benefits of this approach and how it is used in an object oriented world.

Keyword: *Stepwise refinement, Deductive, Procedural Language,*

1. Introduction

1.1 What is Top-down programming ?

Top-down programming, as the name implies, takes a high level definition of the problem and subdivides it into sub problems, which can then be solved to a pieces that will be easy to code. In order words, to solve a large problem,

- break the problem into several pieces and work on each piece separately;
- to solve each piece, treat it as a new problem that can itself be broken down into smaller problems;
- repeat the process with each new piece until each can be solved directly, without further decomposition [2].The technique for writing a program using top–down methods is to write a main procedure that names all the major functions it will need. Later, the programming team looks at the requirements of each of those functions and the process is repeated. These compartmentalized sub-routines eventually will perform actions so simple they can be easily and concisely coded. When all the various sub-routines have been coded the program is ready for testing. [4].

Top down programming, also called “deductive reasoning or stepwise refinement” is a software development technique that imposes a hierarchical structure on the design of the program. It starts out by defining the solution at the highest level of functionality and breaking it down further and further into small routines that can be easily documented and coded. [5] .

Top-down and bottom up are both strategies of information processing and knowledge ordering, used in a variety of fields including software, humanistic and scientific theories and

management and organization. In practice, they can be seen as a style/design methodology.

A top-down approach, in many cases used as a synonym of analysis or decomposition, since is the breaking down of a system to gain insight into its compositional sub-systems. In a top-down approach an overview of the system is formulated, specifying but not detailing any first-level subsystems. Each subsystem is then refined in yet greater detail, sometimes in many additional subsystem levels, until the entire specification is reduced to base elements. A top-down model is often specified with the assistance of "black boxes", these make it easier to manipulate. However, black boxes may fail to elucidate elementary mechanisms or be detailed enough to realistically validate the model. Top down approach start with the big picture. It breaks down from there into smaller segments. [4]

2. Background

Top-down design was promoted in the 1970s by IBM, researcher Harlan Mills and Niklaus Wirth. Mills developed structural programming concepts for practical use and tested them in a 1969 project to automate the New York Times morgue index. The engineering and management success of this project led to the spread of the top-down approach through IBM and the rest of the computer industry. Among other achievements, Niklaus Wirth, the developer of Pascal programming language, wrote the influential paper *Program Development by Stepwise Refinement*. Since Niklaus Wirth went on to develop languages such as Modula and Oberon (where one could define a module before knowing about the entire program specification), one can infer that top down programming was not strictly what he promoted. Top-down methods were favored in software engineering__until the late 1980s, and object-oriented programming assisted in demonstrating the idea that both aspects of top-down and bottom-up programming could be utilized. [4]

3. Top-Down Design

When designing and implementing something, it is generally preferable to start at the most abstract level, and work down from there. For example, when designing and building a house, we prefer to start by deciding what kind of house we want. This in turn helps us determine how many floors, the rooms on each floor, and eventually what kind of building materials we'll need. This is the top-down approach.

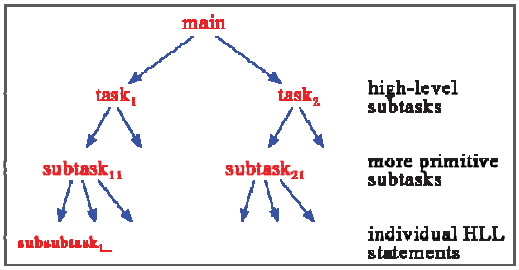
The bottom-up approach begins with the materials we have available and determines the higher level features based on what's possible with what we have. For example, if we start by deciding that our house will be made of snow, mud, or straw, this will strongly influence the type of house we end up with.

The top-down approach doesn't prevent us from using pre-existing components at any level of the design or implementation. It merely allows the design to dictate the components, rather than vice-versa.

These principals apply to computer programs as well as to houses. In order to end up with a program that closely matches our needs, it is preferable to begin at the highest level of abstraction (what the program does and how is used), and let that determine how the more detailed levels are designed and implemented.

In the implementation stage, the top-down approach involves writing the main program first, along with a *stub* for each subprogram called by main. A stub is simply an empty subprogram which has a complete *interface* [6].

.

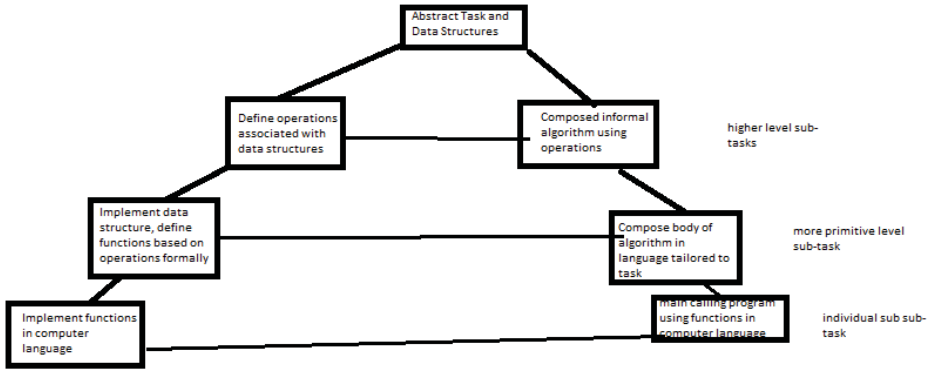


- Describe a problem in terms of high-level sub-problems.
- For each of the high-level sub-problems:
  - Describe it in its turn in terms of slightly simpler sub-problems.
- Continue this breakdown until the subtasks are sufficiently simple to be implemented directly as statements in a programming language.

Each subtask becomes a subprogram (procedure/function).

3.1 Top Down Design in an Object Oriented World

Fig. Functional Decomposition Diagrams



4. The Design Process And Implementation

4.1 Abstract Level

In design, as we have say before, it is preferable to start at the most abstract level, and work down from there. For example, when designing and building a house, we prefer to start by deciding what kind of house we want to build. This in turn helps us determine how many floors, the rooms on each floor, and what kind of building materials we'll need.

4.2 Higher Level Task

Write down the functionality of your system so you have a clear picture of what it does. For example, you can draw out the site map for a website and list all the pages, which will identify all the top-level functional requirements.

The data appearing in our design is processed by means of certain operations. In fact, the particular data structure that one chooses for a given situation depends largely on the frequency with which specific operations are performed. In our top down example below, data structure

operation are how to readIntegers, sortIntegers, printIntegers.

4.3 More primitive level

a) Identify the *actors* in your system (what are the major components in your system?). List the major message and data traffic interaction, which usually highlights some new helper objects.
b) Group similar tasks or aspects of your program into a single object; objects have to play the role of *modules* or *services* in Java.

4.4 Individual sub task

For each method of each object, apply top-down functional decomposition. Each method should have as few "operations" as possible while still being a complete concept. Further break down each of these operations until you think you have an "atomic" operation that is just a few simple programming instructions.

5. Top-Down Example

Write a program to read in a sequence of integers, sort them, and print the sorted sequence.

5.1 Step 1

```
public static void main(String[] args)
    {
    //readIntegers;
    //sortIntegers;
    //printIntegers;
    }
```

5.2 Step 2

readIntegers

- Input?
- Output?
- Lets store the integers in an array.
- Use a loop to read them in.

5.3 Step 3

readIntegers

```
    /**
    * Reads in 100 integers.
    */
    public static int[ ] readIntegers()
    {
    int[ ] vals = new int[100] ;
    // Loop to read in integers
    return vals ;
    }
```

5.4 Step 4

sortIntegers

- Input?
- Output?

```
/**
* Sorts an array of integers.
*/
public static int[] sortIntegers(int[] vals)
{
// Do the sorting
return vals;
}
```

5.5 Step 5

sortIntegers

- Input?
- Output?

```
/**
* Sorts an array of integers.
*/
public static int[] sortIntegers(int[] vals)
{
// Do the sorting
return vals;
}
```

5.6 Step 6

```
public static void main(String[] args)
{
int[] vals = readIntegers() ;
vals = sortIntegers(vals) ;
printIntegers(vals) ;
}
```

5.7 Ways of gradually filling in programs when doing top-down.

- // read in integers
- // TODO – read in integers
- readInIntegers(vals);
- static void readInIntegers(int[] vals){/* TODO – write code */}
- static void readInIntegers(int[] vals){

```
// TODO – replace this dummy code
for(int i;i<vals.length;i++)
vals[i]=i;
}
```

5.8 Actually write the function! A variant using object references

```
public static void main(String[] args)
    {
    int[] myInts = new int[100] ;
    readIntegers(myInts) ;
    sortIntegers(myInts) ;
    printIntegers(myInts) ;
    }
```

[3]

6. Advantages of Top Down Programming

1. Top-down allows the designer to dictate the components to use.
2. Operation and maintenance resources are not initially impacted as severely as with the bottom-up approach.
3. The top level function that calls the sub functions is like an index
4. Each sub function has a nice name that clearly identifies its function.

6.1 Break functions into smaller chunks

It is better to start design with a smaller and simple task, and then proceed into a bigger task because;

1. It is hard to understand long sequences, especially if there are IF conditionals
2. Hard to reuse big chunks whereas each smaller chunk could be reusable; factoring your code into smaller methods usually makes the whole program shorter due to code reuse.

Top-down program design is a useful and often-used approach to problem solving. However, it has limitations:

- It focuses almost entirely on producing the instructions necessary to solve a problem. The design of the data structures is an activity that is just as important but is largely outside of the scope of top-down design.
- It is difficult to reuse work done for other projects. By starting with a particular problem and subdividing it into convenient pieces, top-down program design tends to produce a design that is unique to that problem. Adapting a piece of programming from another project usually involves a lot of effort and time.
- Some problems by their very nature do not fit the model that top-down program design is based upon. Their solution cannot be expressed easily in a particular sequence of instructions. When the order in which instructions are to be executed cannot be determined in advance, easily, a different approach is required. [3]

7. Conclusion

Top-down programming is one way of solving problems. It is useful for small-scale problems or sub-problems. During the design and development of new products or software, designers and engineers uses both bottom-up and top-down approach. In object oriented programming, you commonly subdivide the problem by identifying domain objects (which is a top down step), and refining them, then recombining them into the final program (a bottom up step). Although an understanding of the complete system is very necessary for good design, leading theoretically to a top-down approach, most software projects attempt to make use of existing code to some degree. It does not scale-up because bottom-up approach allows designers to reuse codes.

References

Chukwudi Igbe ph.D,(2002): *General note on fundamental data structures and algorithm,* Department of Computer Science, Imo State University Owerri.

Dr K R Bond (retrieved march 2013): *Structured Programming versus Object-Oriented Programming.*

Lewis D Griffin,(retrieved June 2013) *10-Top-Down Programming* , Department of computer science UCL

Top down and bottom up design,(retrieved march 2013) en.wikipedia.org/wiki/topdown+bottomup +design. http://en.wikibooks.org/wiki/topdown

Definition of top down ;(10/03/13) http;//www.pcmag.com/encyclopedia/index/

Top down design in an object oriented world ;(10/03/13) en.wikipedia.org/wiki/topdown+bottomup +design. http://en.wikibooks.org/wiki/topdown