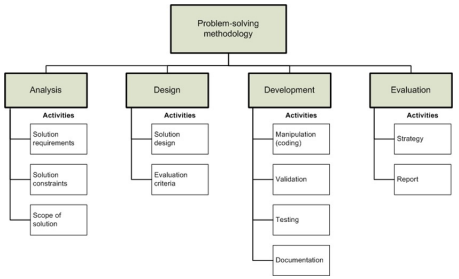
CSE 101
Slide Set 16

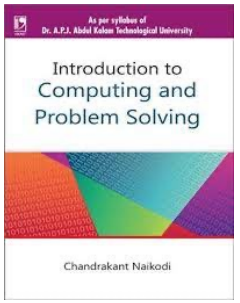
Doç. Dr. Mehmet Göktürk
Department of Computer Engineering

1

Problem Solving Strategies

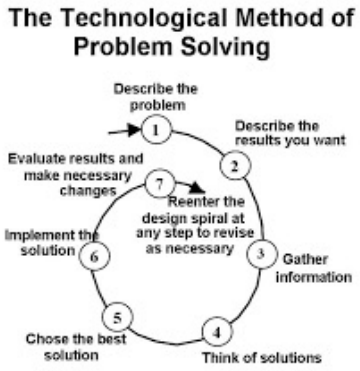
- What is Problem Solving?





2

Iterative Design



3

Problem Solving

Understand and Analyze the Problem :

1. Did you understand the problem fully?
2. Would you be able to explain this question to someone else?
3. What and how many inputs are required?
4. What would be the output for those inputs
5. Do you need to separate out some modules or parts from the problem?
6. Do you have enough information to solve that question? If not then read the question again or clear it to the interviewer.

4

Go Through The Sample Data And Examples Thoroughly

- When you try to understand the problem take some sample inputs and try to analyze the output.
- Taking some sample inputs will help you to understand the problem in a better way.
- You will also get clarity that how many cases your code can handle and what all can be the possible output or output range.

5

Some points:

- Consider some simple inputs or data and analyze the output.
- Consider some complex and bigger input and identify what will be the output and how many cases you need to take for the problem.
- Consider the edge cases as well. Analyze what would be the output if there is no input or if you give some invalid input.

6

Break Down The Problem

- When you see a coding question that is complex or big, instead of being afraid and getting confused that how to solve that question, break down the problem into smaller chunks and then try to solve each part of the problem.
- TOP DOWN design

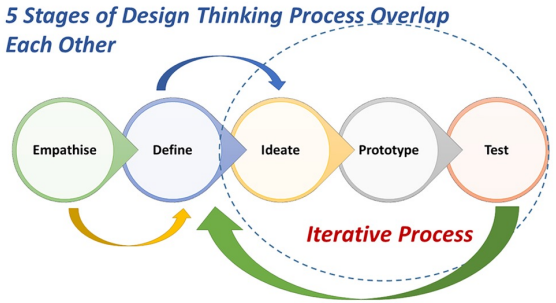
7

Write Pseudocode

- Before you jump into the solution it's always good to write pseudocode for your problem.
- Then replace your pseudocode with real code

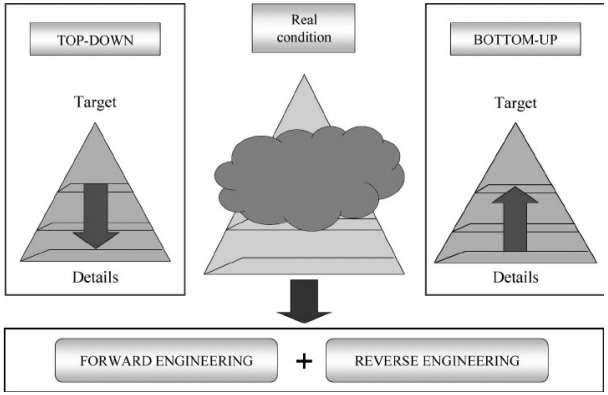
8

Iterative Design



9

Problem Solving



10

Top-Down Design

Topics

- Top-Down Design
- Top-Down Design Examples
- The Function Concept

11

Top-Down Design

- If we look at a problem as a whole, it may seem impossible to solve because it is so complex. Examples:
  - writing a tax computation program
  - writing a word processor
- Complex problems can be solved using **top-down design**, also known as **stepwise refinement**, where
  - We break the problem into parts
  - Then break the parts into parts
  - Soon, each of the parts will be easy to do

12

Advantages of Top-Down Design

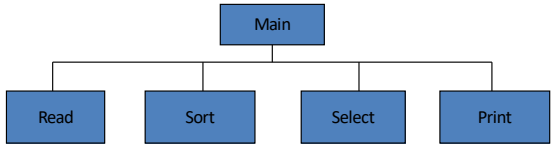
- Breaking the problem into parts helps us to clarify what needs to be done.
- At each step of refinement, the new parts become less complicated and, therefore, easier to figure out.
- Parts of the solution may turn out to be reusable.
- Breaking the problem into parts allows more than one person to work on the solution.

An Example of Top-Down Design

- Problem:
  - We own a home improvement company.
  - We do painting, roofing, and basement waterproofing.
  - A section of town has recently flooded (zip code 21222).
  - We want to send out pamphlets to our customers in that area.

The Top Level

- Get the customer list from a file.
- Sort the list according to zip code.
- Make a new file of only the customers with the zip code 21222 from the sorted customer list.
- Print an envelope for each of these customers.



Another Level?

- Should any of these steps be broken down further? Possibly.
- How do I know? Ask yourself whether or not you could easily write the algorithm for the step. If not, break it down again.
- When you are comfortable with the breakdown, write the pseudocode for each of the steps (**modules**) in the **hierarchy**.
- Typically, each module will be coded as a separate function.

Code Reuse

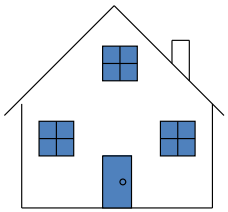
- A primary goal of software engineering is to write error-free code. *Code reuse*, reusing program fragments that have already been written and tested whenever possible, is one way to accomplish this goal.
- Those fragments are written as functions and can be used and reused just like the printf( ) function that you have already been using.

Structured Programs

- We will use top-down design for all remaining programming projects.
- This is the standard way of writing programs.
- Programs produced using this method and using only the three kinds of control structures, sequential, selection and repetition, are called **structured programs**.
- Structured programs are easier to test, modify, and are also easier for other programmers to understand.

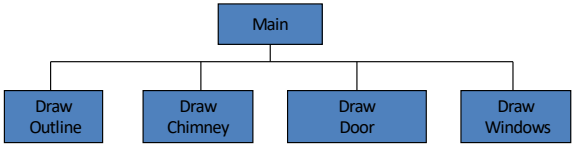
Another Example

- Problem: Write a program that draws this picture of a house.



The Top Level

- Draw the outline of the house
- Draw the chimney
- Draw the door
- Draw the windows



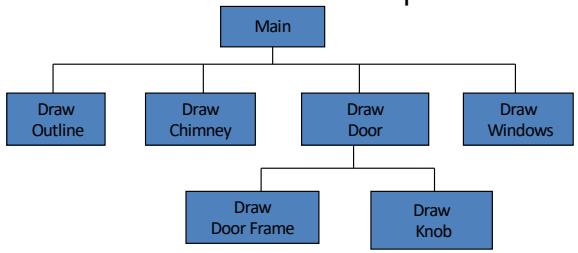
Pseudocode for Main

Call Draw Outline
Call Draw Chimney
Call Draw Door
Call Draw Windows

21

Observation

- The door has both a frame and knob. We could break this into two steps.



22

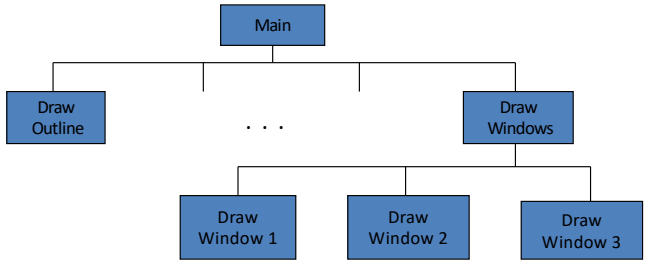
Pseudocode for Draw Door

Call Draw Door Frame
Call Draw Knob

23

Another Observation

- There are three windows to be drawn.

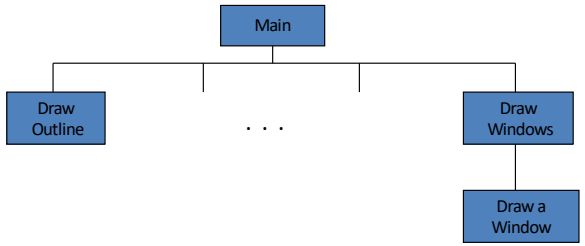


24

One Last Observation

- But don't the windows look the same? They just have different locations.
- So, we can reuse the code that draws a window.
  - Simply copy the code three times and edit it to place the window in the correct location, or
  - Use the code three times, "sending it" the correct location each time (we will see how to do this later).
- This is an example of **code reuse**.

Reusing the Window Code



Pseudocode for Draw Windows

Call Draw a Window, sending in Location 1

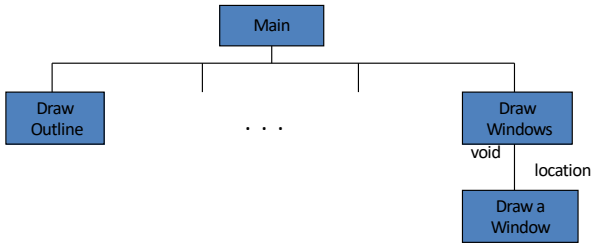
Call Draw a Window, sending in Location 2

Call Draw a Window, sending in Location 3

Pseudocode for Draw Windows (cont'd)

- In the case of the Draw Windows function, we have to supplement (or augment) the information with the specific location each time we use that function. This is a very common thing to do.
- On our structure, we can add the information about the argument(s) to help document the design. We can also add the return information, if any.

Reusing the Window Code - 2



Writing The Code

- Now we can write the shell of the program:

```
#include <stdio.h>

int main( void )
{
   return 0;
}
```


Writing The Code (cont'd)

- Looking at the structure chart, we can now design the prototypes:

```
void drawAWindow( int xCoord, int yCoord);
```

- Now we can add it to the shell of our program.

Writing The Code (cont'd)

- Now we can write the shell of the program:

```
#include <stdio.h>
void drawAWindow( int xCoord, int yCoord);
int main( void )
{
   return 0;
}
```

Writing The Code (cont'd)

- Using the prototype, we can add the shell of the function to the program.
- We would do the same thing for each function in our structure chart.
- Structure of the program is:
  - #include
  - #define
  - Prototypes
  - Functions (It does not matter which function comes first, but normally, people put main( ) first.

Writing The Code (cont'd)

```
#include <stdio.h>
void drawAWindow( int xCoord, int yCoord);
int main( void )
{
   return 0;
}
void drawAWindow( int xCoord, int yCoord)
{
}
```

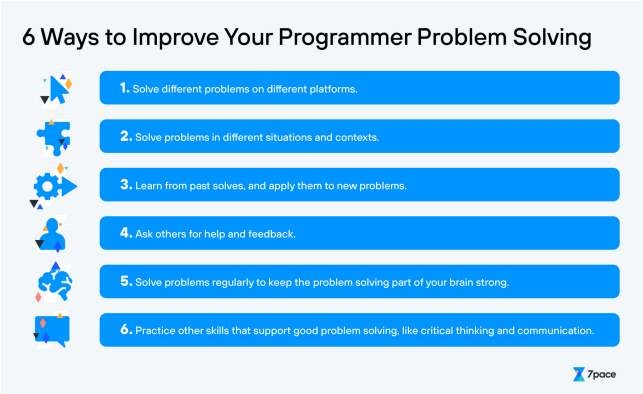

Writing The Code (cont'd)

- Now we can see that the program would really be more useful and give a better solution, if all of our functions were given the the location for each thing that is to be drawn.
- Remember the general solution is the best solution

Good Programming Practices Review

- Use top-down design.
- Use Incremental programming.
- Reuse code.
- Use stepwise refinement.
- Use structured programming:
  - Sequential
  - Selection
  - Repetition

Helpful tricks



37

Simplify and Optimize your Code

- Always try to improve your code. Look back, analyze it once again and try to find a better or alternate solution.
- Explore the problem completely with all possible solutions and then write down the most efficient or optimized solution for your code.
  - Does this code run for every possible input including the edge cases.
  - Is there an alternate solution for the same problem?
  - Is the code efficient? Can it be more efficient or can the performance be improved?
  - How else can you make the code more readable?
  - Are there any more extra steps or functions you can take out?
  - Is there any repetition in your code? Take it out.

38

Questions?

39