IT2205: Programming I

Section 4
Computer program design (05 hrs)

Introduction to program design

INTRODUCTION

Most of us like to *do* things more than we like *planning* the doing. We would rather take a vacation than read travel books about a vacation. That tendency leads to all kinds of complications. If you try planting a new hedge without first spacing the plants, you will inevitably end up with extra plants and no room to plant them or no plants and lots of room to plant them. There are many examples and plenty of cartoons based on the results of poor planning. Picture in your mind the popular photo of two sets of railroad tracks approaching the center for each side and not quite lining up.

The urge to jump in and do something appears almost irresistible around computers. New users of application programs avoid manuals like the plague, preferring to work it out “hands on.” Computer programmers are famous for fiddling with a program until that approach is that it takes longer, usually leads to round-about solutions, and probably leaves undiscovered logic errors in the programe.

INTRODUCTION Cont...

The urge to jump in and do something appears almost irresistible around computers. New users of application programs avoid manuals like the plague, preferring to work it out “hands on.” Computer programmers are famous for fiddling with a program until that approach is that it takes longer, usually leads to round-about solutions, and probably leaves undiscovered logic errors in the programe. It is programming by reacting to crisis ^crisis being defined here as a discovered “bug” or error in the program). Instead of planning in advance to avoid the were going to build a house, even a doghouse, you wouldn’t just start nailing boards together. You would work out what you wanted the completed house to look like and plan the steps needed to accomplish the goal. Lack of planning results in problems like a doghouse that is built in the basement and is too big to fit through the door to the backyard, a hot tub added to a deck that hasn’t been reinforced to hold the extra weight, or a fireplace added without a chimney.

INTRODUCTION Cont...

This book is about avoiding crises by doing the planning. You will study how to plan out program solutions to problems. Since the book is really about the planning, you will never actually implement the solutions. We will never write (code) the programs and run them on the computer. Coding and executing a program is one way to test to see if a program will work, but it’s a pretty inefficient way. You wouldn’t think much of an architect who designed a building, but couldn’t tell you if the building would stand up without actually building it. We need to be able to test our program logic without having to “build” the program. This book will cover designing the logic solutions to a variety of programming problems and testing those solutions to be sure they will work.

Program Development Cycle

Every program goes through the same development process. You may use different terms or descriptions, but you will still follow these steps.

Review the specifications

Informal Design

- List major tasks
- List subtasks, sub-subtasks, and so on Formal Design

Formal Design

- Create formal design from task lists
- Desk check design

Code and compile the program

Test and, if necessary, debug the program

Use and, as necessary, maintain the program

What is a program?

A program is a set of instructions written so the computer can follow them. If we want a computer to calculate what the pay should be for each employee this week, we would need to give the computer two different types of information.

First we tell it all the steps that must be completed to calculate the payroll. This might include calculating a gross pay figure, calculating multiple deductions, and subtracting the deductions from the gross pay figure. In addition the computer needs to know if it should repeat all the defined steps for a second employee, and, assuming it should, when it should stop repeating the steps. All these steps are included in the program.

What is a program? Cont...

The second type of information we must give the computer are the values it should use in the calculations. To calculate the gross pay, the computer needs to know the hourly pay rate and the number of hours worked. To write the paycheck, the computer needs to know the name and Social Security number of the employee. This information forms the input data.

Let's look at each of these steps in detail.

What is a program? Cont…

REVIEW THE SPECIFICATIONS

Some need has been presented to you that has initiated the program creation. Programs always grow out of a need for some form of information, which is another way of saying that we need some form of output from the computer.

The specifications may be very specific and define exactly what the output should look like, which columns certain data should go in, what should be double spaced, and so on. Or the “specs” may be a very casual, oral request (“Hey, could you give me a list of students eligible for the Dean’s List?”) without any formatting requirements.

When you review the specs, ask yourself

(1) do you understand what is required?

(2) do you have sufficient instructions to carry it out? (do you know the eligibility requirements for the Dean’s List?) and

(3) do you have the necessary input available to generate the desired output?

What is a program? Cont...

INFORMAL DESIGN

Don't expect to be able to just sit down and write the design. You need to experiment with a variety of possible solutions and then decide on a general approach, before you formalize those ideas in a design. The informal design step lets you make changes in your plan before you invest too much time in one approach. There are two steps in the informal design that need to be completed frequently in a repeated cycle.

- List the Major Tasks - What are the three to seven major tasks that need to be taken to satisfy the specs? Most people have trouble getting “major” enough.

What is a program? Cont…

INFORMAL DESIGN Cont…

Don’t get caught listing the detail tasks, you’re not ready for the Dean’s List, your major tasks might be
(1) read a record,
(2) check the record against the Dean’s List criteria, and
(3) write the record if the criteria are met.
You would repeat the tasks for each student in the file. Don’t worry here about where you’re going to write it, how you’re going to read it, or what the criteria actually are.

- List the Subtasks – Now go back to each major task, one at a time, and list the subtasks for each major task. The advantage with this approach is that you only need to concentrate on one major task at a time. Now is the time to dig down into some of the specifics. For each major task, you may have up to six or seven subtasks, and each subtask may have additional sub-subtasks under it. You are creating an outline of WHAT needs to be done in our program.

What is a program? Cont...

Once you have completed the major task and subtasks lists, you should figuratively step back a bit, reread the specifications, and make sure you have met all the requirements in the specs. Now is the simplest time to make modifications to your lists of WHAT. You can easily recorder your subtasks, change the break-down into major tasks, and so on. You are not committed to the first attempt, and frequently the first attempt will not be the best solution. Play with your lists a bit until you are satisfied that you have included everything in a "logical" approach. Keep in mind that there is no single, right answer.

What is a program? Cont...

FORMAL DESIGN – The formal design takes your lists and puts them into a recognized format that others could easily read and use. Before an architect does the formal plans, time is spent with sketches and rough ideas.

Only when those have evolved into a coherent concept will the architect invest the time into a formal building plan. The rough sketches probably wouldn't be enough for a builder or another architect to know what is planned. The blueprints, or formal design, utilize a standard form for communicating the plan.

What is a program? Cont...

You’re taking the same approach. Your informal design (task lists) are the best time to “play around” with possible solutions. When you convert them into a formal design, you are investing your time into putting the solution into a complete design that others could easily follow. The task lists defined WHAT needed to be done, the formal design adds HOW it will be done.

- Create formal Design From Task Lists – Just as the blueprint is a standard communication tool for architects and builders, the formal design is a communication tool for program designers and coders. We need a design format that will be understandable by other designers (because we frequently design programs in teams with each team member working on a part of the whole program) and by coders (because they will actually translate in into code).

What is a program? Cont...

In addition, eventually your formal design will become a part of the program's documentation package that will stay with the program as long as it is in use. There are many possible design tools, but this book will concentrate on two of them. Flowcharts and pseudo code. Both are somewhat standard means for communicatig the design steps. Each has its advantages (and its disadvantages). In the next chapter, we will look at flowcharting as a design tool, and the chapter following it introduces pseudo code.

- Desk Check the Design – Just as the architect doesn't want to have to build the house to see if it meets the specifications, you don't want to have to code your program to check it. Desk checking is the way to check the logic of your design. You will work through the design, step by step, implementing each step as the computer would, "reading" the test date, doing any calculations, and "writing" output.

What is a program? Cont...

Desk checking will tell you if there are any logic errors (assuming you have created good test data) before you invest the time in coding. What if you have errors? Go back to the task lists, make the necessary changes, implement those changes in the formal design, and try the desk checking again. The Informal Design and Formal Design are really a cycle that you perform until your desk checking tells you the logic is perfect. The first example of desk checking is in Chapter 2 with the first formal design. We will desk check some of the designs throughout the book, but you should get in the habit of desk checking *all* of your designs.

CODE AND COMPILE THE PROGRAM- Once you know that you have a good design, coding the program is simply translating each design statement into the proper syntax for the desired programming language.

The End of Section