Paradigms of Programming

Procedural Code

Example Languages: C, BASIC, Pascal

# Key Features

* Computational steps are split into procedures.
* Hierarchical call of procedures.

# Description

The procedural paradigm is typified by the hierarchical structuring of computation as a series of calls. Top-level procedures delegate work to lower level procedures. The work is passed as arguments to lower-level procedures and those procedures return their results as return values, modified argument values or state changes in global data structures.

void printReport(List<String> reportItems)

{

  printReportHeader();

  printReportBody(reportItems);

  printReportFooter();

}

# A bit more about it...

One of the challenges in the procedural paradigm is the crafting of appropriate data structures. Since argument passing between computational levels is the primary mechanism for communicating work across a procedural program, we must be aware of where data is needed in each part of our program and how we might group it to avoid exposing it to areas of the program which do not need it. Much of our work is focused on tuning these structures and arriving at procedures with evocative names and clear inputs and outputs.

The following example illustrates some of the tradeoffs involved in the procedural approach:

void processInvoices(List<Invoice> invoices, PaymentSchedule schedule, TaxCalculator calculator)

{

  ...

  TaxTable table = produceAdjustedTable(schedule, calculator);

  ...

}

The function produceAdjustedTable requires PaymentSchedule and TaxCalculator. We could consider bundling them together into a common data structure so that we do not have to pass two arguments to produceAdjustTable. If we do, we'd be able to reduce the number of arguments to processInvoices also. However, we should scan through the rest of the program to discover whether they are other data structures which would be better grouped with either PaymentSchedule or TaxCalculator. If there are, those groupings may be more appropriate.

In languages with function pointers (or their equivalent) the procedural paradigm leads naturally into object orientation. Data structures can contain references to behavior, and with that change, much of the work involved in tuning data to make it "work" for the various functions it will be passed to goes away.

The procedural paradigm is, however, is useful in many places at smaller scales.