Bottom-up programming

What is bottom-up programming

Bottom-up programming is the opposite of top-down programming. It refers to a style of programming where an application is constructed starting with existing primitives of the programming language, and constructing gradually more and more complicated features, until the all of the application has been written.

Advantages of bottom-up programming

Bottom-up programming has several advantages over top-down programming .

Testing is simplified since no stubs are needed. While it might be necessary to write test functions, these are simpler to write than stubs, and sometimes not necessary at all, in particular if one uses an interactive programming environment such as Common Lisp or GDB.

Pieces of programs written bottom-up tend to be more general, and thus more reusable, than pieces of programs written top-down. In fact, one can argue that the purpose bottom-up programming is to create an application-specific language . Such a language is suitable for implementing an entire class of applications, not only the one that is to be written. This fact greatly simplifies maintenance, in particular adding new features to the application. It also makes it possible to delay the final decision concerning the exact functionality of the application. Being able to delay this decision makes it less likely that the client has changed his or her mind between the establishment of the specifications of the application and its implementation.

How does bottom-up programming work?

In a language such as C or Java, bottom-up programming takes the form of constructing abstract data types from primitives of the language or from existing abstract data types.

In Common Lisp, in addition to constructing abstract data types, it is common to build *functions* bottom-up from simpler functions, and to use macros to construct new *special forms* from simpler ones.

One may ask why it is not possible to construct functions and special forms bottom-up in other languages than Common Lisp. Constructing functions bottom-up requires a way of passing complicated arguments between functions. Common Lisp uses *lists* for such argument passing. Lists are flexible standardized data structures in the language. In other languages, data structures would have to be defined for such parameter passing only, making it more like an abstract data type than just a function. With respect to special forms, only the two-level syntax of Common Lisp allows a flexible enough macro facility for bottom-up programming of special forms.