

COMP205
Comparative Programming
Languages

Part 1: Introduction to programming languages

Lecture 3: Managing and reducing complexity, program processing

MANAGING AND REDUCING COMPLEXITY, AND PROGRAM PROCESSING

1. **Managing and reducing complexity**
 - Problem decomposition
 - Abstraction
 - Contextual checking (strong and heuristic type checking)
2. **Programming processing**
 - Interpretation v Compilation
 - Libraries
 - Macro processing
 - Debugging tools
 - Program Management Systems and Environments

MANAGING AND REDUCING COMPLEXITY

Three techniques:

- Decomposition – Problem subdivision.
- Abstraction – Ignoring irrelevant detail.
- Contextual checking – internal consistency checking.

A good programming language supports all three!

PROBLEM DECOMPOSITION

- Divide and conquer (divide et impera).
- Problem decomposition hinges on procedures, recursion and parameter passing, and can be applied in most (high level) programming languages.

ABSTRACTION

Ignoring irrelevant detail in a safe way (information hiding).

Requires the use of an “interface” to abstract away from lower level detail.

To do this safely users should have no choice but to abstract away.

Abstraction is typically facilitated through the use of *packages* or *modules*.

CONTEXTUAL CHECKING

- Contextual checking is concerned with the contextual correctness of program code.
- Ideally we would like to check for (and eradicate) all possible “run time” errors, however contextual checking is a difficult undertaking and in some cases (e.g. recursion) completely impractical.
- Contextual checking consists (typically) of parameter and identifier validation.

STRONG TYPE CHECKING

- The most successful contextual checking technique to date is *strong type checking*.
- Many programming languages require that the *type* of a data items are specified.
- Strong type checking is then concerned with the analysis of a program to ensure that *all* data items are of the correct type.
- The alternative is *weak type checking*.

HEURISTIC TYPE CHECKING

- Heuristic contextual checking is concerned with the application of “rules of thumb”.
- For example the rule of thumb that all functions must end with a “return statement”, or that all recursive definitions must have at least one base case.

PROGRAM PROCESSING

- A program written in a high level language (*source code*) can only be run in its machine code equivalent format.
- There are two ways of achieving this:
 1. Compilation, and
 2. Interpretation.

INTERPRETATION

- Interpretation requires the use of a special program that reads and reacts to source code.
- Such a program is called an *interpreter*.
- During interpretation run-time errors may be detected and “meaningful” error messages produced.

COMPILATION

- Compilation requires the use of a special program (called a *compiler*) that translates source code into *object* code.
- Sometimes the object code cannot be directly executed. Various library files must be “linked in” using another special program called a *linker*, which produces executable code.
- Again various contextual checks are made during compilation.

LIBRARIES

- Libraries (in computer programming terms) contain chunks of precompiled (object) code for various functions and procedures that come with a programming language that requires compilation.
- For example functions and procedures to facilitate I/O.

INTERPRETATION VERSUS COMPILATION

Interpretation	Compilation
Slow	Fast
Good error messaging	Poor error messaging

MACRO PREPROCESSING

- A *macro* comprises a name and a string.
- During macro preprocessing all occurrences of the name within the program are replaced by the string before interpretation/compilation takes place.
- Use of macros offers the advantage of enhanced readability.
- However it is also argued that a well designed language should not require the use of macros!

DEBUGGING TOOLS

- To assist in error detection many debugging tools exist.
- Some of these allow the user to analyse the *core dump* that occurs in the event of a fatal error. (A core dump describes the “state” of a program when a fatal error occurs).
- Others allow programmers to step through and execute a program line by line to support analysis of its execution.
- Overall modern debugging is still in a very unsatisfactory state of development.

PROGRAM MANAGEMENT SYSTEMS

- Program development comprises a code-writing (using a text editor) and compilation loop, as the programmer attempts to eradicate all detected errors.
- In a large program, comprising many modules, it makes sense to recompile only those modules that have been changed.
- The job of a program manager is to monitor which modules require recompilation.
- The management system is usually integrated with the linker.

PROGRAMMING ENVIRONMENTS

- To further reduce this development time some vendors have combined a text editor with a compiler/interpreter into a single dedicated programming environment for the production of code in a particular programming language.
- Such environments include a program management systems and other “administrative” tools (e.g. version control).

SUMMARY

1. Managing and reducing complexity

- Problem decomposition
- Abstraction
- Contextual checking

2. Programming processing

- Interpretation v Compilation
- Libraries
- Macro processing
- Debugging tools
- Program Management Systems and Environments