COMP3031: Principles of Programming Languages

Lecture 1: Introduction

Qiong Luo

Department of Computer Science and Engineering
Hong Kong University of Science and Technology

What's a programming language for?

Stroustrup (creator of C++) gave the following possible answers in 1994:

- A tool for instructing machines?
- A means for communicating between programmers?
- A vehicle for expressing high level designs?
- A notation for algorithms?
- A way of expressing relationships between concepts?
- A tool for experimentation?
- A means for controlling computerized devices?
- A collection of "neat" features?

Which ones are correct? Stroustrup's answer: All of the above except the last one.

Why study this course?

- Improve your understanding of the language you are using.
- Increase your vocabulary of useful programming concepts and constructs.
- Make it easier to learn a new language.
- Make it easier to design a new language.
- Allow a better choice of programming language.

Let's start with a brief history.

Machine language is unintelligible

- A code fragment for ENIAC, the original von Neumann machine in 1946:

```
00000010101111001010
00000010111111001000
00000011001110101000
```

  It adds the numbers in location 10 an 11 and stores the result in location 12.

- Goldstine (ENIAC programmer, 1972): "While this enumeration is virtually unintelligible to a human, it is exactly what the machine *understands*."

Assembly language is low level

- It is a variant of machine language in which names and symbols take the place of the actual codes for machine operations, values, and storage locations.
- M[i] := M[j]+M[k] puts the sum of the values in location j and k into location i.
- It is low level because it accesses the actual memory locations, relies on the actual instruction set of a particular machine, therefore is highly machine dependent.

FORTRAN, the first higher level PL

Fortran (*For*mula *Tran*slation), the first PL (1957):

- A = B + C
- A = B**2 - 4.0*A*C
- A, B, C are *symbols* denoting values.
- There is no direct reference of memory location.
- Machine independent operations.

Benefits of higher-level languages

- Readable familiar notations.
- Machine independent (portability).
- Availability of program libraries.
- Consistency checks during implementation that can detect errors (good compilers, and programming environment).

Next we briefly overview major programming paradigms of higher-level PLs.

Imperative programming

- Action oriented. A computation is a sequence of actions. Primitive actions are machine-oriented: assignments, read, write, *etc.*
- Begins with Fortran, matures in Algol 60. Representatives: Pascal (a teaching language) and C (a language for low-level system programming).
- An example in C:

```
#include <stdio.h>
main()
{ int nc;
  nc = 0;
  while (getchar() != EOF)
   ++nc;
  printf(``%d\n'',nc);
}
/* count characters in input */
```

Object-oriented programming (OOP)

- Ways of organizing and structuring programs.
- Decomposition of a task into subtasks.
- Subtasks only talk to each other through a pre-defined interface.
- A subtask, by default, has no access to (nor does it care about) the details of how the other subtasks are implemented.
- The notions of classes, subclasses, and inheritance from a class to its subclasses are central to OOP.
- Begins with Simula (1969), a language designed for simulation domain. Example: A traffic simulation can have many cars and many trucks. A natural way of describing this domain would be to have a subclass of cars and a subclass of trucks, both of which inherit from a class of vehicles.
- Languages support OOP include Smalltalk, C++, Java, and others.

Functional Programming (FP)

- A program implements a mapping: from input to output values.
- In imperative programming, this mapping is achieved indirectly: by commands that read inputs, manipulate them, and write output.
- In FP, this mapping is achieved directly: the program is a function, typically composed from simpler ones.
- Begins with Lisp (List Processor, 1958), a language designed for applications in Artificial Intelligence.
- FP languages include Lisp family (Lisp, Mac Lisp, Scheme, Common Lisp), and ML (MetaLanguage).
- An example in ML:

```
fun square x = x*x;
val square = fn : int -> int
square 5;
val it = 25 : int
```

Logic Programming (LP)

- Begins with Prolog (1972), a language based on mathematical logic and first used for natural language processing:

```
Every psychiatrist is a person.
Every person he analyzes is sick.
Jack is a psychiatrist in Hong Kong.

Is Jack a person?
Where is Jack?
Is Jack sick?
```

- Prolog has been used for applications ranging from deductive databases to expert systems.

Implementation of PL: Bridging the Gap

There are at least two ways a language can be implmented:

Compilation: translates a program into code that can be run directly on the computer. Programs in imperative programming languages like Fortran, Pascal, and C are normally compiled. For example, the Fortran expression B**2 - 4.0*A*C may be translated into a sequence of assembly instructions:

```
R1 := B
R1 := R1 * R1
R2 := 4.0
R3 := A
R2 := R2 * R3
R3 := C
R2 := R2 * R3
R1 := R1 - R2
```

Bridging the Gap (Cont'd)

Interpretation: There is a virtual computer called interpreter for the PL. A program in that PL runs on the interpreter. Programs in FP languages like Lisp and ML, and in Prolog are often interpreted.

```
function eval(E) =
 IF E is the constant i THEN i
 ELSE IF E is the sum of E1 and E2
      THEN eval(E1)+eval(E2)
 ELSE IF E is the product of E1 and E2
      THEN eval(E1)*eval(E2)
```

Main Points

- A higher-level PL is required in order to solve many complex problems using a computer.
- There are hundreds of these languages, under different paradigms.
- We are going to study some basic concepts and constructs underlying most of them: formal grammars and procedural calls mechanisms.
- You are already familiar with imperative programming and object-oriented programming.
- You will learn two more in this course: functional programming and logic programming.