Issues of programming languages
Expressions and control

6.1-6.6

Expressions and Control

- Expressions
  - Assignment
  - Others
- Control

A model of the hardware of computer

- The hardware has two critical components (in terms of computation)
  - Memory
  - CPU
- What can a CPU do?
  - ***Move*** data from one memory location to another
  - ***Arithmetic and logic*** operations
    - +, -, *, /
    - Compare two values (equal, smaller etc.), bitwise conjunction
  - ***Control flow***
    - Sequential execution of instructions (stored in the memory)
    - Jump to another instruction (stored in the memory)

A model of computing

- Church-Turing hypothesis: informally, any algorithms can be represented by a finitely terminated Turing machine
- Turing machine

Formally, a Turing machine is a 7-tuple,

$$(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}}),$$

where Q is the (finite) set of states, Σ is the input alphabet such that Σ does not contain the special **blank** symbol $\sqcup$, Γ is the tape alphabet such that $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$,

$$\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$$

is the transition function, $q_0 \in Q$ is the start state, $q_{\text{accept}} \in Q$ is the accept state, and $q_{\text{reject}} \in Q$ is the reject state, where $q_{\text{accept}} \neq q_{\text{reject}}$.

Other models of computing

- Lambda calculus – leading to functional programming languages
- …

Programming languages

- Grows from machine language (and mathematics)
  - Memory unit address and its content vs. names
  - Arithmetic and logic operations of CPU vs. expressions in a language
  - Move data from location to another location vs. assignment
  - Jump vs various control structures

Expressions

- From the point view of applications, mathematics is widely used to model problems, so, it is desirable for a programming language to represent some mathematical expressions
  - Arithmetic expressions
  - Logical expressions

- Expressions – precedence and associativity
  - As a designer, usually you need to give the rules on them
  - As a programmer, you want to understand such rules defined by the designers

Assignment

- X := X + 1
- Assignment never appeared before (at least in mathematics). It is invented by computer scientists.
- Its meaning: place a value to a memory location.

Model of variables and assignment

- To define the meaning of assignment, there are usually two approaches
  - Value model of variables
  - Reference model of variables

- Value model of variables
  - Variables on the left side of an assignment (called *l-value*) denote references (memory locations/addresses). Variables on the right side of an assignment (called *r-value*) denote values
  - Example
    - a := 2; b := a;

- Reference model of variables
  - Variables on both sides denote references

```
public class point{
    public int x;
    public int y;
    ……
}
point a, b;
a := new point;
b := a;
a.x := 10;
```

- Orthogonality means that features can be used in any combination and the meaning is consistent regardless of the surrounding features

```
if (a=b) {…}

a := begin f(b); g(c) end;
```

Initialization of variables

- Why?
  - Accidental use of uninitialized variable is one of the most common programming errors.

Control

- Sequencing (supported by computer, natural in applications)
- Selection / condition. (partially supported by machine language. natural in application)
- Iteration. (Reducing complexity, natural in application)
- Procedural abstraction
- Recursion
- Concurrency
- Nondeterminacy

Sequencing

- Read 6.3

Selection

- Read 6.4.
- switch case
  - Not only more natural; could be faster in implementation (using indexed table)
- Note short-circuited conditions
  - (a > b) & (b > c) [when a>b is false, no need to evaluate b>c]
  - (a > b) | (b > c) [when a > b is true, …]

Goto and more structured control

- In machine language, we have “jump” to change the order of execution of instructions
- However, programming languages avoid using “jump”
  - Jump leads to programs too complex for human being to prove that the program is correct. More structured control is used to reduce the complexity.
  - In the solution of many applications, it is natural to have conditions and iterations. So, more structured control makes programming easier.

Iterations

- Enumeration-controlled loops
  - For each value in a given set, loop is executed once.
  - E.g., for loop in C/C++, Java, …
  - for(i=start; i <= last; i+=step) {}
  - for i in range(start, last, step) {}
  - break / continue may be used to jump out of a loop or jump to the start of the loop
- A logically controlled loops
  - The loop is executed until some condition changes

Iteration

- Iteration is sufficient to represent any algorithms, i.e., making the language Turing complete.
- So, without goto, we don’t lose any power of the language

Enumeration-controlled loops

- Enumeration-controlled loops

  FOR i:=first TO last BY step DO

  …

  END

  i: *index* of the loop;

  first: *initial* value of index variable

  last: *bound* of index variable

- Translation into goto (jump) /

```
            r1 := first;
            r2 := step;
            r3 := last;
L1:         if r1 > r3          goto L2
            …
            r1 := r1 + r2
            goto L1
L2:
```

- Why enumeration loop
  - We have this phenomenon in many applications
  - It has a clear/simple meaning that helps assure the correctness of a program
- Note “goto” can do this. But, just “can do” is not good enough. We need effectiveness in achieving things: ease of solving application problems and assurance of correctness

- Design issues
  - Can control enter or leave the loop in a way other than the one provided by the loop?
    - No.
    - Most languages allows break/exist to the next instruction after the loop
    - Fortran 77 allow to jump into the body of a loop: bad idea

- What happens when the loop body modifies the variables used in computing bound?
  - Most languages specify that the bound is computed only once before the first iteration
- What happens if the loop body modifies the index variable itself?
  - Many languages prohibit changing index

– Can the program use the index after the loop ends? if so, what's the value of the index?
  - The first one exceeds the bound?
  - The bound?
  - Fortan 90 and Pascal: says it is not defined, i.e., a programmer should not use it.

Enumeration -- iterators

- An abstract problem: we may wish to iterate over the elements of any sets
- In python

  for i in range(first, last, step):

    ……

  range: *iterator* (built-in) *yields* integers {first, first + step, …}

  The procedure to produce the index is clearly separated from the loop body that is to process these indexes.

- In object oriented language
  - *Iterator*: an object which contains a set of elements. It must have the following methods e.g.,
    - in C++
      - first(): returns the first element in this set
      - next(): returns the next element after the current
      - last(): returns the last element in the set
    - in Java: hasnext(), next(), iterator() – return the iterator (object) itself
  - To enumerate the elements of object X

```
for(iterator i = X.first(); i != X.last(); i=X.next()) { ……}
```

Logically controlled loops

- Loop is dependent on a condition (logical).
- Design issues: when to test the condition?
  - Before the loop (pre-test loops)

    while *condition* do *statement*
  - After the loop (post-test loops)

    do *statement* while *condition*

    repeat *statement* until *condition*
  - In the middle of the loop (midtest loops)

```
for( ; ; ) {
……
if condition break;
……
}
```

Combination loops

- Algol 60 provides a single loop construct that subsumes modern enumeration and logically controlled loops.
- In C we have

```
for (i = first; i <= last; i += step) {
……
// break / continue is allowed
……
}
```

Recursion

- Recursion: a function is allowed to call it self in its body
- A language with recursion but without iterations is as powerful as a language with iterations but without recursions.
- Most imperative languages allow both. For some applications iteration is more natural while others, recursion is more natural

- Tail recursion
  - In the body of the definition of a recursive function, there is nothing done after the recursive call, i.e., the recursion call is at the tail of this function.
  - There is very efficient implementation for tail recursion!

Summary

- Assignment and other mathematical and logic expressions are basic building blocks of an imperative programming languages
  - Value model vs reference model of variables
- Controls are needed on these building blocks
  - More structured constructs (condition, iterations) vs goto
  - Iteration is “equivalent” to recursion
  - Enumeration loops involve interesting language design decisions