Chapter One

Programming Languages
Recommended Course Textbooks


Programming Languages Overview

• Purpose:
  - Discover language design successes and failures.
  - Discover how languages are designed and implemented.

• Several real languages will be programmed but:
  - Course is not intended to teach programming
  - Experience the key programming language elements that are common to or distinguish two classes of languages.
  - Assume you can already program in at least one object oriented language that uses C++ style syntax.
  - Java used as a recent object oriented, threaded and networking language.
  - Haskell used as a very high-level language mainly for studying and implementing interpretive language.
Programming Languages Overview

• Computer systems have come to effective use with programs consisting of a well-ordered and finite set of non-ambiguous and computable operations.

• A program written in a natural language (English, German, Turkish) can’t be executed from a computer:
  - we need a formal language. It must be a language provided with a set of rules in order to avoid any possible ambiguity.

• Programming languages is basically developed to utilize the capabilities of a computer system.

• A programming language is the problem solving tool of computer science used for human expression of computer solutions.
  - Ideas are expressed in a language.
Programming Languages Overview

• The Sapir-Whorf linguistic theory states that the structure of language defines the boundaries of thought.
  ❑ New ideas often require new language, for example algebra.
• A given language can impede or facilitate certain modes of thought.
• All programming languages are capable of solving any computable problem – computer languages are equivalent.
  ❑ No programming language can prevent a problem solution.
  ❑ A given language can subtly influence the class of solutions examined and the quality of a program.
Some of the Problems

• Hardware simulator to interpret machine language (Java).
• Defining language syntax and parsing (XML).
• Interpreter for high-level language (Hugs).
• Light-weight threads (Java)
• Networking (Java)
• Object-oriented programming (Java)
• Components (Java)
The von Neumann architecture

The main problem of the early computers was the difference in the implementation of data and programs (external connections, wires...) In 1950 J. von Neumann, defined the first computer with memorized program: the EDVAC. It weighted 8 tons and had a memory of 1024 words but it was not so different from our modern PC...

- Interpretation of the instructions, guide of the execution, coordination of the other hardware components
- Storage of data (input/intermediate/output) and programs
- Arithmetical and logical operations
The EDVAC

Electronic Discrete Variable Automatic Computer, which has the following features

- 6,000 vacuum tubes
- 12,000 diodes
- 56 kW of power consumption
- 45 m$^2$ of floor space
- 8 tons of weight
The modern personal computers

• They basically follow the scheme given by von Neumann: they are only much more powerful in computation and storage capabilities

1) Not only the central memory (RAM) but also the hard disk (permanent data storage)

2) Several input devices: keyboard, mouse, CD, DVD, scanner, microphone, camera, joystick...

3) Several output devices: screen, printer, speakers, plotter...
Chapter Outline

• What makes programming languages an worthwhile subject of study?
  o The amazing variety
  o The odd controversies
  o The intriguing evolution
  o The connection to programming practice
  o The many other connections
The Amazing Variety

• There are very many, very different languages
• A list of computer languages (~2500) is provided on
  http://people.ku.edu/~nkinners/LangList/Extras/langlist.html
• *Hello world* programs for many languages:
  http://en.wikipedia.org/wiki/Hello_world_program
• Often grouped into four families:
  • Imperative
  • Functional
  • Logic
  • Object-oriented
A Sample Language List
Imperative Languages

• Example: a factorial function in C

```c
int fact(int n) {
    int sofar = 1;
    while (n>0) sofar *= n--;    
    return sofar;
}
```

• Hallmarks of imperative languages:
  - Assignment and side-effects
  - Iteration
  - Order of execution is critical
Functional Languages

• Example: a factorial function in Haskell

    fact x =
        if x <= 0 then 1 else x * fact(x-1);

• Hallmarks of functional languages:
  - Single-valued variables
  - Heavy use of recursion
  - Functions are first-class citizens, can be used as parameters, function results, etc.
  - Minimal use of assignments and side-effects
Another Functional Language

• Example: a factorial function in Lisp

\[
\text{(defun fact (x)}
\text{ \hspace{1cm} (if (<= x 0) 1 \hspace{1cm} (* x (fact (- x 1))))})
\]

• Looks very different from Haskell
  □ Fully-parenthesized, prefix syntax

• But Haskell and Lisp are closely related
  □ Single-valued variables: no assignment
  □ Heavy use of recursion: no iteration
Logic Languages

• Example: a factorial function in Prolog

```prolog
fact(X,1) :-
    X =:= 1.

fact(X,Fact) :-
    X > 1,
    NewX is X - 1,
    fact(NewX,NF),
    Fact is X * NF.
```

• Hallmark of logic languages
  - Program expressed as rules in formal logic
  - Execution attempts to prove a result based upon rules
Object-Oriented Languages

• Example: a Java definition for a kind of object that can store an integer and compute its factorial

• Hallmarks of object-oriented languages:
  - Usually imperative, plus…
  - Constructs to help programmers use “objects”—little bundles of data that know how to do things to themselves
public class MyInt {
    private int value;
    public MyInt(int value) {
        this.value = value;
    }
    public int getValue() {
        return value;
    }
    public MyInt getFact() {
        return new MyInt(fact(value));
    }
    private int fact(int n) {
        if (n <= 0) return 1;
        else return n * fact(n-1);
    }
}

Strengths and Weaknesses

• The different language groups show to advantage on different kinds of problems

• Decide for yourself at the end of the semester, after experimenting with them

• For now, one comment: don’t jump to conclusions based on factorial!
  - Functional languages do well on such functions
  - Imperative languages, a bit less well
  - Logic languages, considerably less well
  - Object-oriented languages need larger examples
About Those Families

• There are many other language family terms (not exhaustive and sometimes overlapping)
  ☐ Applicative, concurrent, constraint, declarative, definitional, procedural, scripting, single-assignment, …

• Some languages straddle families

• Others are so unique that assigning them to a family is pointless
Example: Forth Factorial

: FACTORIAL
    1 SWAP BEGIN ?DUP WHILE TUCK * SWAP 1- REPEAT ;

• A stack-oriented language
• Postscript language used by printers is similar
• Could be called imperative, but has little in common with most imperative languages
Example: APL Factorial

\( \times / \iota X \)

- An APL expression that computes X’s factorial
- Expands X it into a vector of the integers 1..X, then multiplies them all together
- (You would not really do it that way in APL, since there is a predefined factorial operator: !X)
- Could be called functional, but has little in common with most functional languages
Outline

• What makes programming languages an interesting subject?
  - The amazing variety
  - The odd controversies
  - The intriguing evolution
  - The connection to programming practice
  - The many other connections
The Odd Controversies

• Programming languages are the subject of many heated debates:
  - Partisan arguments
  - Language standards
  - Fundamental definitions
Language Partisans

• There is a lot of argument about the relative merits of different languages
• Every language has partisans, who praise it in extreme terms and defend it against all detractors
• To experience some of this, explore newsgroups: comp.lang.*
• (Plenty of rational discussion there too!)
Language Standards

• The documents that define language standards are often drafted by international committees
  - Can be a slow, complicated and rancorous process
  - Fortran 82 8X 88 90 standard released in 1991
Basic Definitions

• Some terms refer to fuzzy concepts: all those language family names, for example

• No problem; just remember they are fuzzy
  ❑ Bad: Is X really an object-oriented language?
  ❑ Good: What aspects of X support an object-oriented style of programming?

• Some crisp concepts have conflicting terminology: one person’s argument is another person’s actual parameter
Outline

• What makes programming languages an interesting subject?
  - The amazing variety
  - The odd controversies
  - The intriguing evolution
  - The connection to programming practice
  - The many other connections
The Intriguing Evolution

- Programming languages are evolving rapidly
  - New languages are being invented
  - Old ones are developing new dialects
New Languages

• A clean slate: no need to maintain compatibility with an existing body of code
• But never entirely *new* any more: always using ideas from earlier designs
• Some become widely used, others do not
• Whether widely used or not, they can serve as a source of ideas for the next generation
Widely Used: Java

- Quick rise to popularity since 1995 release
- Java uses many ideas from C++, plus some from Mesa, Modula, and other languages
- C++ uses most of C and extends it with ideas from Simula 67, Ada, Clu, ML and Algol 68
- C was derived from B, which was derived from BCPL, which was derived from CPL, which was derived from Algol 60
Not Widely Used: Algol

- One of the earliest languages: Algol 58, Algol 60, Algol 68
- Never widely used
- Introduced many ideas that were used in later languages, including
  - Block structure and scope
  - Recursive functions
  - Parameter passing by value
Dialects

• Experience with languages reveals their design weaknesses and leads to new dialects
• New ideas pass into new dialects of old languages
Some Dialects Of Fortran

- Original Fortran, IBM
- Major standards:
  - Fortran II
  - Fortran III
  - Fortran IV
  - Fortran 66
  - Fortran 77
  - Fortran 90
  - Fortran 95
  - Fortran 200x?

- Deviations in each implementation
- Parallel processing
  - HPF
  - Fortran M
  - Vienna Fortran
- And many more…
Outline

• What makes programming languages an interesting subject?
  - The amazing variety
  - The odd controversies
  - The intriguing evolution
  - The connection to programming practice
  - The many other connections
The Connection To Programming Practice

• Languages influence programming practice
  - A language favors a particular programming style—a particular approach to algorithmic problem-solving

• Programming experience influences language design
Language Influences
Programming Practice

• Languages often strongly favor a particular style of programming
  - Object-oriented languages: a style making heavy use of objects
  - Functional languages: a style using many small side-effect-free functions
  - Logic languages: a style using searches in a logically-defined problem space
Fighting the Language

• Languages favor a particular style, but do not force the programmer to follow it
• It is always possible to write in a style not favored by the language
• It is not usually a good idea…
  - C++ is not good for logic programming.
  - Prolog is not good for systems programming.
Imperative Haskell

Haskell makes it impossible to use assignment and side-effects. However some Haskell code seems imperative:

```
fact n =
    foldl (*) 1 [1..n]
```

OR

```
fact n =
    foldl (\t m->t*m) 1 [1..n]
```
Non-object-oriented Java

Java, more than C++, tries to encourage you to adopt an object-oriented mode. But you can still put your whole program into static methods of a single class:

class Fubar {
    public static void main (String[] args) {
        // whole program here!
    }
}
Functional Pascal

Any imperative language that supports recursion can be used as a functional language:

```pascal
function ForLoop(Low, High: Integer): Boolean;
begin
  if Low <= High then
  begin
    {for-loop body here}
    ForLoop := ForLoop(Low+1, High)
  end
  else
  begin
    ForLoop := True
  end;
end;
```
Programming Experience Influences Language Design

• Corrections to design problems make future dialects, as already noted

• Programming styles can emerge before there is a language that supports them
  □ Programming with objects predates object-oriented languages
  □ Automated theorem proving predates logic languages
Obfuscatable Languages

- **Obfuscated code** is source code that is very hard to read and understand, often intentionally. Some languages are more prone to obfuscation than others.
- Sample obfuscatable languages
  - C, C++, Perl, …
- There are programming contests which reward the most creatively obfuscated code
  - International Obfuscated C Code Contest
  - Obfuscated Perl Contest
Sample C Code (1)

```c
(____,____,____){____/__<=1?(____,____+1,____):!(____%____)?(____,____+1,0):____%____==____/____&!(____)?(printf("%d\t",____/____),(____,____+1,0)):(____%____>1&!(____%____<____/____)?(____,1+____,____+!(____/____%(____%____))):(____%____<____*____?____,____+1,____):0;}main(){(100,0,0);}
```

- The above code prints out the prime numbers less than 100, which is equivalent to:

```c
f (a,b,c) {
    b/a<=1 ? f(a,b+1,c)
    : !(b%a) ? f(a,b+1,0)
    : b%a==b/a & !c ? (printf("%d\t",b/a),f(a,b+1,0))
    : b%a>1 & b%a<b/a ? f(a,1+b,c+!(b/a%(b%a)))
    : b<a*a ? f(a,b+1,c)
    : 0;
}
```

```c
main() {
    f (100, 0, 0);
}
```

Output:

```
2 3 5 7 11 13 17
19 23 29 31 37 41
43 47 53 59 61 67
71 73 79 83 89 97
```
Another C Code (2)

```c
main(l, a, n, d) char**a; {
    for (d = atoi(a[1]) / 10 * 80 - atoi(a[2]) / 5 - 596; n = "@NKA\nCLCCGZAAQBEAAD AfISADJABBA\nSNLGAQABDAXIMBAACTBATAHDBAN\ZcEMMCCCCAAhEIJFAEAAABAfHJE\TbdFLDAANEnfDNBPPhDbcBBBEA_AL"[l++-3];) for (; n --> 64;)
        putchar(!d++ + 33 ^ l & 1);
}
```
Another C Code (2)

if the program is compiled from `map.c` into `map`

Output: map 36 42
Outline

• What makes programming languages an interesting subject?
  □ The amazing variety
  □ The odd controversies
  □ The intriguing evolution
  □ The connection to programming practice
  □ The many other connections
Other Connections: Computer Architecture

• Language evolution drives and is driven by hardware evolution:
  ☐ Call-stack support – languages with recursion
  ☐ Parallel architectures – parallel languages
  ☐ Internet – Java
Other Connections: Theory of Formal Languages

• Theory of formal languages is a core mathematical area of computer science
  ❑ Regular grammars, finite-state automata – lexical structure of programming languages, scanner in a compiler
  ❑ Context-free grammars, pushdown automata – phrase-level structure of programming languages, parser in a compiler
  ❑ Turing machines – Turing-equivalence of programming languages
Turing Equivalence

• Languages have different strengths, but fundamentally they all have the same power
  \[ \{ \text{problems solvable in Java} \} = \{ \text{problems solvable in Fortran} \} = \ldots \]

• And all have the same power as various mathematical models of computation
  \[ \{ \text{problems solvable by Turing machine} \} = \{ \text{problems solvable by lambda calculus} \} = \ldots \]

• *Church-Turing thesis*: this is what “computability” means
Conclusion

- Why programming languages are worth studying (and this course worth taking):
  - The amazing variety
  - The odd controversies
  - The intriguing evolution
  - The connection to programming practice
  - The many other connections
- Plus…there is the fun of learning a new language!