CS 210
Programming Languages

1

Course description

- This course presents major features of programming languages, with primary emphasis on the role of particular language features in writing good software; programming language design alternatives; various programming paradigms embodied in languages, such as procedural, data-flow, functional and object-oriented languages.

2

Student Outcomes

- Upon completing this course, students will
  - learn various programming languages' strengths and weaknesses.
  - learn basics of lexical analysis, parsing, semantic analysis, and error handling from a language implementer's point of view.
  - understand several programming language paradigms, and have a feel for what kinds of problems are best solved in each language paradigm.

3

3

Student Outcomes

- Upon completing this course, students will
  - have initial experience with formal languages, automata and grammars.
  - have experience with lexical analyzers and parsers and with one or more non-procedural languages (e.g. ML, Lisp, Prolog, Icon, etc.)

4

4

Course calendar

- Lectures (02/20/2023 ~ 06/16/2023):
  - W: 8:40 am – 10:10 am (Group 1), 10:30 am – 12:00 pm (Group 2)
  - Th (starts from 05/03/2023):
    1:30 pm – 3:00 pm (Group 1), 3:20 pm – 4:50 pm (Group 2)
- Midterm exam:
  - 05/11/2023 (tentative)
- Final exam:
  - TBD, 06/19/2023 ~ 06/23/2023

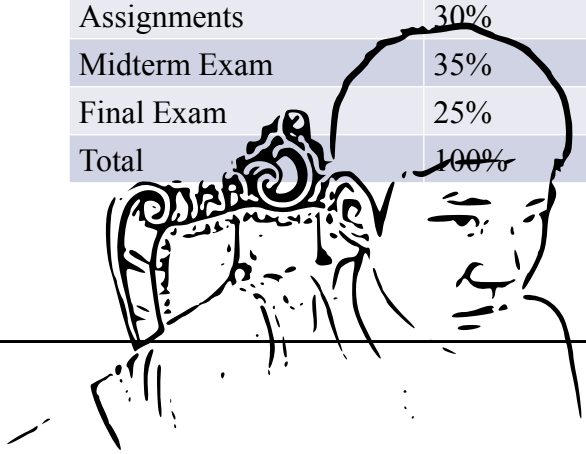
5

5

Grading

- Grading system

Total	Percentage
Attendance/Participation	10%
Assignments	30%
Midterm Exam	35%
Final Exam	25%
Total	100%



6

6

Students requirements

- Please check QQ frequently.
- Attendance is taken during each lecture. Should a student need to miss any lecture, assignment, exam, etc., 班主任 must be consulted beforehand.
- Unexcused absence from lectures, assignments or exams will lead to no credit and will not be made up.
- All assignments will be submitted to 课代表 while exams are required to be turned in via email to the professor directly.

7

7

Students requirements

- Students are expected to turn in all assignments on time. No late work will be accepted.
- Students must show all steps in the answers (homework assignments and exams) and highlight or box your final answers.
- All exams are closed-book and closed-notes. The University operates a zero tolerance policy in relation to cheating in examinations.
- Using electronic devices such as laptops, tablets, cellphones, etc. is prohibited during exams.

8

8

Academic Honesty

- While teamwork and team-learning are recommended throughout the semester, no form of plagiarism will be tolerated. Cheating, fabrication, plagiarism or helping others to commit these acts will not be tolerated.
- Academic dishonesty will result in disciplinary action including, but not limited to, failure of the student assessment item or course, and/or dismissal from the University.
- Please refer to your Student Handbook or ask your 班主任 for more information.

Chapter 1
Programming Languages

The Amazing Variety

- There are very many, very different languages
- (A list that used to be posted occasionally on `comp.lang.misc` had over 2300 published languages in 1995)
- Often grouped into four families:
  - Imperative
  - Functional
  - Logic
  - Object-oriented

Imperative Languages

- Example: a factorial function in C

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

- Hallmarks of imperative languages:
  - Assignment
  - Iteration
  - Order of execution is critical

Functional Languages

- Example: a factorial function in ML

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

- Hallmarks of functional languages:
  - Single-valued variables: no assignment
  - Heavy use of recursion: no iteration

13

13

Another Functional Language

- Example: a factorial function in Lisp

```
(defun fact (x)
  (if (<= x 0) 1 (* x (fact (- x 1)))))
```

- Looks very different from ML
- But ML and Lisp are closely related
  - Single-valued variables: no assignment
  - Heavy use of recursion: no iteration

14

14

Logic Languages

- Example: a factorial function in Prolog

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

```
% The factorial of X is 1 if
% X equals 1 and
% cut predicate, similar to "break"
% The factorial of X is Fact if
% X > 1 and
% NewX is one less than X and
% the factorial of NewX is NF and
% Fact is X  times NF
```

- Hallmark of logic languages
  - Program expressed as rules in formal logic

15

15

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

16

16

Object-Oriented Languages

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

17

17

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

18

18

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 *multi-paradigm* languages straddle families: JavaScript, OCaml, Python, Ruby
- Others are so unique that assigning them to a family is pointless

19

19

Example: Forth

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

- A stack-oriented language
- Postscript is similar
- Could be called *imperative*, but has little in common with most imperative languages

20

20

Example: APL

$$\times / \iota X$$

- 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

21

21

The Odd Controversies

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

22

22

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

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
- How to define object-oriented languages?
- 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?

25

25

The Intriguing Evolution

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

26

26

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

*Basic Combined Programming Language

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

29

29

Dialects

- Experience with languages reveals their design weaknesses and leads to new dialects
- New ideas pass into new dialects of old languages
- Original Fortran, IBM
- Major standards:
  - Fortran II
  - Fortran III
  - Fortran IV
  - Fortran 66
  - …
- Deviations in each implementation
- Parallel processing
  - HPF
  - Fortran M
  - Vienna Fortran
- And many more…

30

30

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
  - E.g. recursion and conditional expressions were introduced to Lisp.
  - The designer needed them in AI applications.

31

31

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

32

32

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…

Imperative ML

ML makes it hard to use assignment and side-effects. But it is still possible:

```
fun fact n =
  let
    val i = ref 1;
    val xn = ref n
  in
    while !xn>1 do (
      i := !i * !xn;
      xn := !xn - 1
    );
    !i
  end;
```

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:

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

37

37

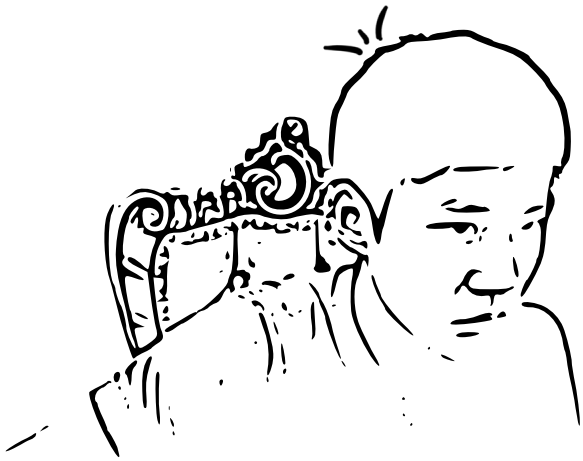
Other Connections: Computer Architecture

- Language evolution drives and is driven by hardware evolution:
  - Call-stack support – languages with recursion
  - Parallel architectures – parallel languages
  - Internet – Java

38

38

Questions?



39

39