PLAN FOR TODAY

History of Programming Language Families
Procedural Languages
Object Oriented Languages
Logical Languages
Functional Languages

PROGRAMMING LANGUAGE FAMILIES
Procedural, Object Oriented, Logical, and Functional Languages

EARLY BEGINNINGS

Beginnings of automatic computation - Charles Baggage's Difference
First electronic computation engine - ENIAC
Conditional Control Transfer - John Von Neumann
First Compiler – A-o by Dr. Grace Murray

HISTORY

Year | Language
--- | ---
1952-53 | SHORTCUT, Speedcoding, Lancing & Zierer
1953 | FORTRAN
1955 | FLOW-MATIC
1955-60 | COBOL, JOVIAL
1958 | ALGOL 58 (IAL)
1959-60 | COBOL, JOVIAL, LISP
1960 | AIDOL 60
1961 | GPDL, SIMSCRIPT
1963 | SNOBOL, SHOOL
1964 | JOSJ, BASIC, PL/I, FORTRAN
1965 | AFL, SIMULA 67
1966 | AIDOL 6R
1970 | Pascal
1972 | C, PROLOG
1974 | Ada
1980 | Smalltalk-80

HISTORY OF PROGRAMMING LANGUAGE FAMILIES

- FORTRAN – originally imperative language, evolved into a procedural language
- Logical programming with languages such as Prolog
- Hope – First functional programming language
- Object oriented began in 1960s
- Considered paradigm that absorbs all other paradigms

PROCEDURAL

Nick
**PROcedural Programming Languages**

- Traditional, widely used paradigm
- C, PASCAL, C++, Java
- Defined set of instructions
- “main function” calls other functions/procedures
- Modularity, Scoping

**PROCEDURAL PROGRAMMING: MODULARITY**

- Procedures act independently
- Easy to group procedures
- Modules can call use other modules
- Intuitive organization of program

**PROCEDURAL PROGRAMMING: SCOPING**

- Required data given on procedure call
- Can be bypassed by using global data
- Data hidden from other procedures
- C example:
  ```c
  int mult (int x, int y)
  {
    return x * y;
  }
  ```

**ADVANTAGES AND DISADVANTAGES**

**Pros:**
- Simple and easy to learn
- Logical ordering easy to understand
- Can solve most problems

**Cons:**
- Harder to solve complex problems compared to OOL
- Solutions often require increased amount of complicated code compared to OOL

**OBJECT Oriented Programming Languages**

- Programs are organized as cooperative collections of objects
- Techniques and structure that a programmer use for object orientation
PROCEDURAL VS. OBJECT-ORIENTED

'ACTION', 'PROCEDURE', 'METHOD'

WHAT TO DO, NOT ON 'HOW TO DO'

Withdraw, deposit, transfer

Customer, money, account

OBJECTS AND CLASSES

Classes reflect concepts, objects reflect instances that embody those concepts.

Variable → object

Type → class

Daria
Jane
Brittany

OBJECTS AND CLASSES

• Classes reflect concepts, objects reflect instances that embody those concepts.

TECHNIQUES

- Data abstraction → represents essential features without including background details and explanations
- Encapsulation → hiding an object's data
- Inheritance → to inherit the variables and methods of an existing class
- Polymorphism → create objects of different types

ADVANTAGES AND DISADVANTAGES

- Shorter development times
- Easier code sharing
- Flexibility

• Learning and getting used to how OOL works may not be as easy as it seems
• More time to understand and implement object-oriented code
• Not all software engineering problems can be solved or made easier using OOL.

Inheritance

Polymorphism
LOGIC PROGRAMMING LANGUAGES

- Falls under the category of Declarative Programming
- Comes with some great advantages. Supports:
  - Non-Determinism
  - Pattern-Matching
  - Meta-Programming
- Although it comes with neat features, it is not an easy language to master.

Non-Determinism
A non-deterministic language is one that can support, at various points in a program, different alternatives for program flow.

Pattern-Matching
As the name suggests, pattern-matching is the act of checking for a sequence.

Meta-Programming
Meta-programming is writing computer code that can manipulate, or write other programs, or even itself.

query: ?- sibling(sally, erica) !

1. To prove this query, Prolog first looks through the facts and rules, called clauses.
2. Matching clause is sibling(X, Y) :- parent_child(Z, X), parent_child(Z, Y).
3. Proving this query means proving the body.
4. Substitute sally for X to get parent_child(Z, sally).
5. To prove above clause, Prolog looks through the clauses, and there is the parent_child(X, Y) rule, which is true if either father_child(X, Y) or mother_child(X, Y) is true. A choice point is created (non-determinism).
6. First check father_child(X, Y) by substituting Z, sally to father_child(Z, X), creating father_child(Z, sally).
7. Again look through the clauses. father_child(Z, sally) can be proven with the father_child(tom, sally) clause if Z is substituted with tom.
8. First part of the body is proven, and above steps are repeated with parent_child(tom, erica).

FUNCTIONAL PROGRAMMING VS. IMPERATIVE PROGRAMING

Most programing is imperative
Functional used mainly in academia
Hardware optimized for imperative
Can be more productive using functional

FUNCTIONAL

Stephen

FIBONACCI IN C (IMPERATIVE)

#include <iostream>

// Fibonacci numbers, imperative style
int fibonacci(int iterations) {
  int first = 0, second = 1; // seed values
  for (int i = 0; i < iterations - 1; ++i) {
    int sum = first + second;
    first = second;
    second = sum;
  }
  return first;
}

int main() {
  std::cout << fibonacci(10) << "\n";
  return 0;
}
FIBONACCI IN HASKELL (3 LINES)

-- Fibonacci numbers, functional style
-- describe an infinite list based on the recurrence relation
for Fibonacci numbers fibRecurrence first = first : fibRecurrence second (first + second)
-- describe fibonacci list as fibRecurrence with initial values 0 and 1
fibonacci = fibRecurrence 0 1
-- describe action to print the 10th element of the fibonacci list
main = print (fibonacci !! 10)

FIBONACCI IN HASKELL (1 LINE)

fibonacci2 = 0:1:zipwith (+) fibonacci2 (tail fibonacci2)

KEY FUNCTIONAL PROGRAMMING CONCEPTS

higher order functions
pure functions
recursion
strict vs. non strict
type inference

QUESTIONS?

We got answers...
...Well maybe

REFERENCES

...Well maybe

We got answers!


REFERENCES...