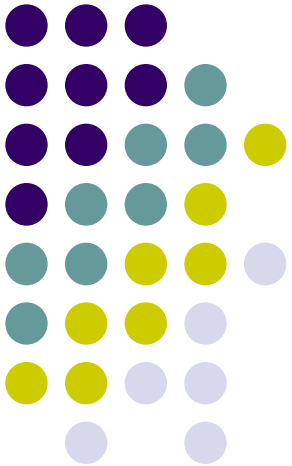
CS 460

Programming Languages

Fall 2023

Dr. Watts

(28 August 2023)



LARC

Need help with a class? Want to take charge of your learning?

The **Learning and Academic Resource Center (LARC)** can help with…

Writing Tutoring

- Brainstorm ideas and get those first paragraphs written!
- Refine your thesis statement or research question
- Help with ANY writing task, not just English courses

Subject-Specific Tutoring

- Study for an upcoming midterm
- Work on challenging assignments
- Develop better study guides
- Gain subject-specific learning skills

Academic Success Skills

- Avoiding procrastination
- Time management
- Study Skills
- Setting and achieving goals

You can find us…

Through the SSU Portal!

Click on our tile to make an appointment!

LARC-WC Online

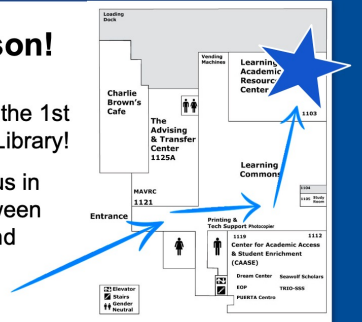
On our Website!

Type **larc.sonoma.edu** into your browser
or
scan this QR code to be automatically directed to our website!

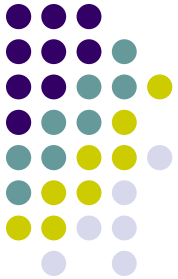
In-Person!

LARC is on the 1st floor of the Library!

Come visit us in person between 9-5 M-Th and 9-4 F!

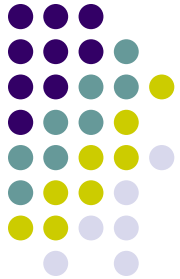


Exercise 1 B



```
========== Compiling Exercise 1 for  /home/faculty/tiawatts/cs460/Ex1/student1 ==========


====== Completed compiling Exercise 1 for  /home/faculty/tiawatts/cs460/Ex1/student1 ======

========== Executing test of Exercise 1 for  /home/faculty/tiawatts/cs460/Ex1/student1 ==========

========== Listing differences for Exercise 1 test run  1  ==========
========== Lines preceded by < are from your output ==========
========== Lines preceded by > are from the expected output ==========

3c3
< 1 $5 bills
---
> 1 $5 bill
5,7c5,7
< 1 50-cent coins
< 1 25-cent coins
< 1 10-cent coins
---
> 1 50-cent coin
> 1 25-cent coin
> 1 10-cent coin
17c17
< 1 1-cent coins
---
> 1 1-cent coin

======= End of differences for Exercise 1 test run  1  =======
```

Course Administration



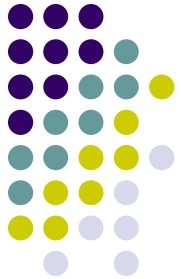
- Survey
- Course website
  http://watts.cs.sonoma.edu/cs460f23/

- BASIC
- FORTRAN
- Pascal
- COBOL
- BPL
- Audit Reporter
- RPG
- JCL
- SNOBOL
- APL

- ALGOL
- BAL
- SAS
- SPSS
- Ada
- LISP
- C
- Logo
- QBasic
- C++

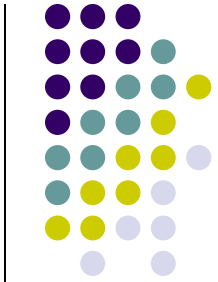
- MFC
- HTML
- Scheme
- Java
- Action Script
- C#
- XNA
- Objective C
- SVG
- Python

Why do we study Programming Languages?



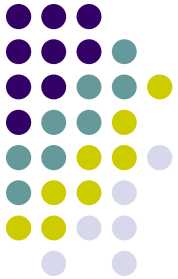
- Choosing languages
- Learning languages
- Efficient program implementation
- Designing and implementing new languages
- Expressing ideas
- Overall understanding

Influences on Language Design



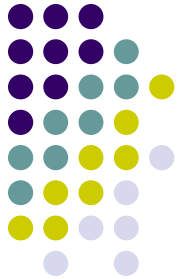
- Architectures
- Domains
- Paradigms

Programming Domains



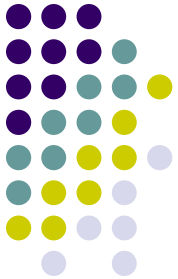
- Science and Mathematics
  - FORTRAN – FORmula TRANslator
- Business
  - COBOL – Common Business Oriented Language
- Education
  - BASIC – Beginners All-purpose Symbolic Instruction Code
- Artificial Intelligence
  - LISP, Scheme
- Systems
  - Assembly languages, C
- Interactive
  - Java, VB, C#
- Web
  - HTML, XML, CSS, SVG

Programming Paradigms



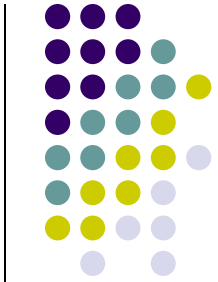
- Procedural
  - FORTRAN, COBOL, BASIC, Pascal
- Functional
  - LISP, Scheme
- Logical
  - Prolog
- Object Oriented
  - Smalltalk, Java
- Scripting
  - RPG, Java Script
- Hybrid
  - C++

Language Design Factors



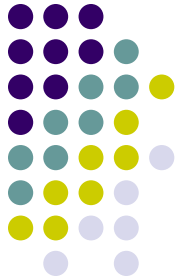
- Readability
- Simplicity
- Orthogonality
- Control Structures
- Data Types/Structures
- Writability
- Reliability
- Cost

Influences on Language Design



- Architectures
  - Single CPU – single processor
  - Single CPU – multiple processors
  - Multiple CPUs
- Domains
  - Calculating devices - ForTran
  - Business applications – COBOL
  - AI - Lisp
  - Education - BASIC
- Paradigms – way in which programs are written
  - Spaghetti code – lots of GOTOs!
  - Structured programming – ALGOL
  - Procedural Programming
  - Object Oriented Programming
  - Functional Programming
  - GUI / Web Programming
  - Parallel Programming

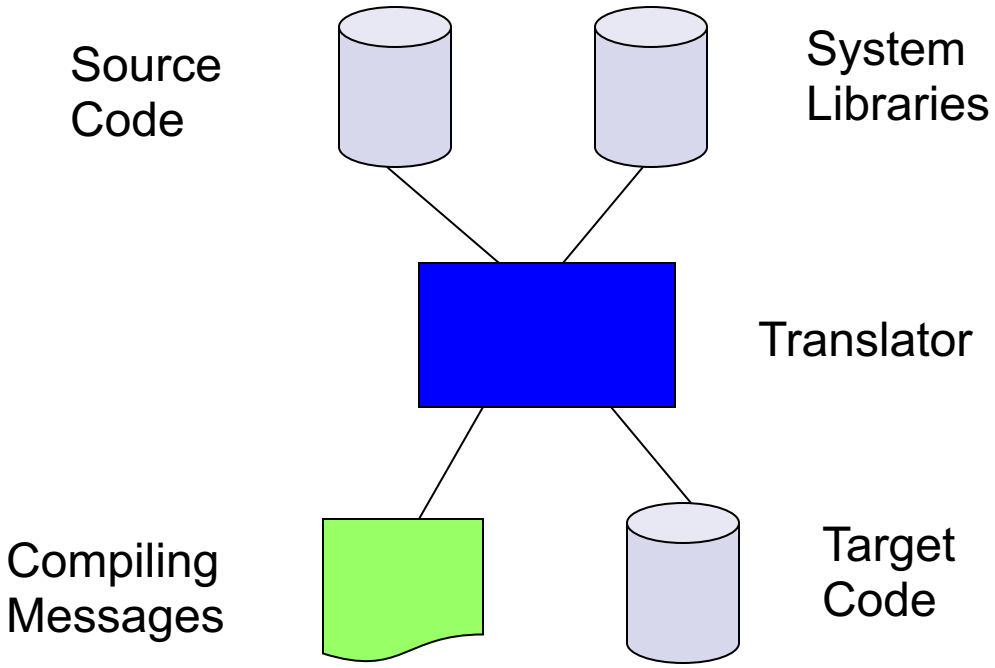
The compilation process



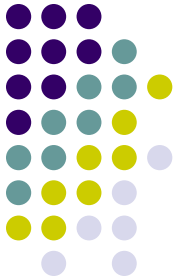
- Input – a human readable source program
  - Text file
  - Conforms to a specific programming language
- Output – a machine readable target program
  - A “binary” file
  - Conforms to a specific machine architecture

Language Translation

System Libraries



Phases of Compilation

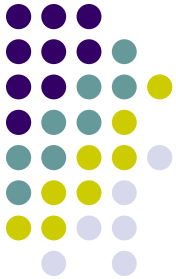


- Lexical analysis
- Syntactical analysis
- Semantic analysis
- Intermediate code generation
- Optimization
- Target code generation

Lexical Analysis

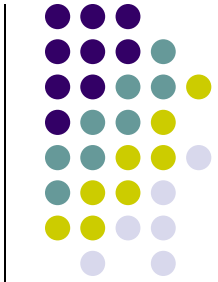
```
int
25.5
;
```

What is a “lexeme”?



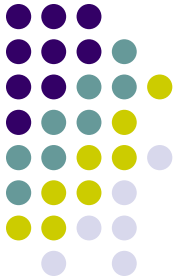
- String of characters with a meaning
- Examples?
  - Key/Reserved words → define display if
  - User defined identifiers → N value v1 def-ine
  - Literals → 12 -12.34 “Hello”
  - Symbols – operators → ( ) ‘ + -
- Defined using regular expressions
- Recognized by the implementation of a DFA

Language Design



- Key (reserved) words (K)
- Symbols (S)
- Literals (L)
- User defined names (U)

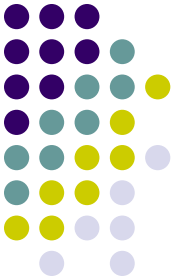
C++ User defined names



- Uses?

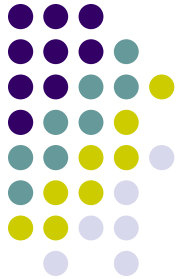
- Rules?

Use of Underscore (_) in User Defined Names



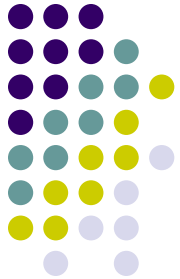
```
#include <iostream>
using namespace std;

int main ()
{
      int _;
      float __;
      string ___;
      char ____;
      bool _____;
      cout<<_<<__<<___<<____<<_____<<endl;
      return 0;
}
```

Regular Expressions



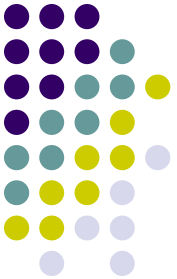
- Alphabet – the symbols that actually appear in the lexeme
- Special symbols to define the regular expression
  - ( ) : grouping
  - * : 0 or more occurrences of a pattern
  - $^{+}$: 1 or more occurrences of a pattern
  - | : indicates alternatives
  - λ : indicates nothing (lambda)

Regular Expression Examples

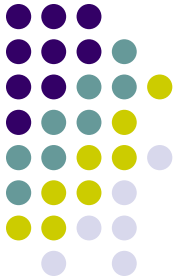


- Alphabet = {a,b,c}
- Examples
  - a (b | c) a →
  - a^+ (b | c) a^+ -->
  - a (b | c)* a →
  - abc*ba →
  - (a|b|c|λ)((ab*c)|(cb*a))$^+$ →

Regular Expression for User Defined Names



A regular expression for unsigned integer numeric literals



A regular expression for signed integer numeric literals