

CS 460 Manual

Fall 2008

Sonoma State University
Computer Science Department
Instructor: Dr. Tia Watts
tia.watts@sonoma.edu

Table of Contents

Table of Contents	2
CS 460 - Course Syllabus – Fall 2008	3
CS 460 - Proposed Course Schedule – Fall 2008	5
Policy on Collaboration	6
Policy on Incomplete Grade.....	6
Fall 2008 University Calendar	7
Tutorial Assignment and Project Submission Instructions.....	9
C Review Exercise 1 (Simple Sorting)	11
C Review Exercise 2 (Advanced Sorting)	13
Scheme Tutorial Exercise 1 (Introduction to Scheme).....	15
Scheme Tutorial Exercise 2 (Scheme Lists)	19
Scheme Tutorial Exercise 3 (Scheme Sorting)	21
Prolog Tutorial Exercise 1 (Intro to Prolog).....	23
Prolog Tutorial Exercise 2 (Prolog “Functions” and Lists)	27
Prolog Tutorial Exercise 3 (Prolog Sorting)	31
C++ Review Exercise 1 (C++ OOP and Sorting).....	33
Java Tutorial Exercise 1 (Introduction to Java)	35
Java Tutorial Exercise 2 (Java Lists and Sorting).....	41
Java Tutorial Exercise 3 (Java Illustration)	45
Chapter 1 Review Questions.....	49
Chapter 2 Review Questions.....	49
Chapter 3 Review Questions.....	50
Chapter 4 Review Questions.....	50
Chapter 5 Review Questions.....	51
Chapter 6 Review Questions.....	52
Chapter 7 Review Questions.....	53
Chapter 8 Review Questions.....	54
Chapter 9 Review Questions.....	55
Chapter 10 Review Questions.....	56
Chapter 1 Review Questions.....	56
Chapter 2 Review Questions.....	56
Chapter 3 Review Questions.....	57
Chapter 14 Review Questions.....	57
Chapter 15 Review Questions.....	57
Chapter 16 Review Questions.....	57
Scheme Review Questions.....	58
Prolog Review Questions.....	59
Java Review Questions	61
Thoughts on Computer Programming Languages	62

CS 460 - Course Syllabus – Fall 2008

Catalog Description

Lecture, 4 hours. A survey of the syntactic, semantic and implementation features of functional, procedural, object-oriented, logic and concurrent programming languages

Prerequisites

CS 252 (CS 250) and CS 315 (CS 254) or instructor consent. Knowledge of C/C++ and data structures are required. Students lacking background in C/C++ will need to familiarize themselves with the language within the first two weeks of the course. Students who have not acquired a strong foundation in the fundamentals of programming and data structures (i.e., earned a grade below a C- in CS 252 (CS 250) and CS 315 (CS 254)) are required to retake the course, or its equivalent, before taking this course.

Instructor

Dr. Tia Watts Darwin 116E (707) 644-2807
e-mail: tiawatts@cs.sonoma.edu (Message subject *MUST* begin with **CS 460**)
Office Hours: Tuesday 1:00 – 2:00 pm Office or Laboratory
Office Hours: Thursday 2:00 – 3:00 pm Office or Laboratory

Class Meetings

Lecture: Tuesday/Thursday 10:00-11:50 am Darwin 29

Course Objectives

Upon successful completion of this course, the student will be able to:

- Describe the steps and concepts required to evaluate a new programming language.
- Describe and implement the steps required to learn a new programming language.
- Describe the highlights of the development of programming languages.
- Describe and differentiate between functional, procedural, object-oriented, and logic programming paradigms.
- Describe the common features of programming languages.
- Describe the implementations of a variety of programming language features.
- Design and implement a recursive descent language translator.

In addition, the student will have used (but not mastered) a variety of programming languages representing different programming paradigms.

Important Dates

26 August 2008	First day of classes
9 September 2008	Last day to ADD or DROP courses
22 September 2008	Last day to WITHDRAW from courses online
16 October 2008	Midterm Exam
11 November 2008	Veteran's Day Observed (No classes – University closed)
26-28 November 2008	Thanksgiving Break (No classes – University closed)
12 December 2008	Last day of classes
16 December 2008	Final Exam: 11:00 am – 12:50 pm

Course Materials

Text Concepts of Programming Languages (8th edition) by Robert W. Sebesta
Addison Wesley Publishing Company, 2008.

Other Materials

CS 460 Course Manual
Loose Leaf Binder; Hole Punch; Small Stapler; Folder

Coursework

Lecture: The proposed outline of the topics to be covered appears in the course schedule. Students are expected to attend all lectures and to read the relevant sections of the text prior to lecture. Students are responsible for making up the missed work if they are absent.

Tutorials: The tutorials for this course are designed to give the student an opportunity to work with programming languages from a variety of programming paradigms. Tutorial exercises must be submitted by the dates indicated in the course schedule.

Programming Projects: A multi-stage programming project will be assigned during the semester. Each stage will require that you write one or more C program modules. Stage due dates and times will be stated on the project assignment handout. Programs must be submitted by the due date and time based on the UNIX date/time stamp on the file(s). No projects will be accepted after the due date and time.

Midterm and Final Exam: The examinations are based on the assumption that each student is responsible for knowing the material covered in class, in the relevant sections of the course text, and the tutorials in the course manual. All exams are closed book; however, one handwritten 8 ½ by 11 sheet of notes (front and back) will be allowed. Exams will include questions related to the programming languages covered in the tutorials. The final exam will be comprehensive. There may be one or more unannounced quizzes given during the semester. Exams cannot be made up unless you have made arrangements with the instructor prior to the date of the exam. Quizzes cannot be made up.

Grading

Homework and Tutorial Exercises	30%
Programming Project	20%
Exams and Quizzes	50%

Grading Scale:

100...	93...	90...	87...	83...	80...	77...	73...	70...	67...	63...	60...	0
A	A-	B+	B	B-	C+	C	C-	D+	D	D-	F	

Note: You must separately earn a passing grade on the tutorial exercises, programming projects, and the exams in order to pass the course.

CS Majors must take this course for a letter grade. University guidelines regarding the grade of Incomplete will be strictly adhered to. Incomplete grades will only be given for circumstances beyond a student's control; inability to keep up with the work due to an excessive course load, for example, is insufficient to warrant an Incomplete. The University also requires that, to be a candidate for an Incomplete grade, the student must currently be doing C (or better) work in the course.

CS 460 - Proposed Course Schedule – Fall 2008

Week	Monday	Lecture Topics	Reading	Tutorial Exercise	Activities
1	25 August	Course Introduction and History of Programming Lang Develop	1,2	C Review Program (Sorting)	
2	1 Sept	Subprograms and recursion	9, 15	C Recursive Sorting	
3	8 Sept	Language translation	3, 4	Intro to Scheme	C Review Exercise due on 9/12 @ 11:59 pm (.c file)
4	15 Sept	Lexical Analysis (scanning)	3, 4	Scheme Lists	Project Part 1 assigned
5	22 Sept	Grammars	3, 4	Scheme Lists and Sorting	
6	29 Sept	Grammars	3, 4	Intro to Prolog	Scheme Tutorial Exercise due on 10/3 @ 11:59 pm (.ss file)
7	6 Oct	Syntactical Analysis (parsing)	3, 4, 16	Prolog Lists	Project Part 2 assigned
8	13 Oct	Scanning & Parsing	3, 4	Prolog Lists and Sorting	Midterm – including C and Scheme (10/16)
9	20 Oct	Names, Typing & Scoping	5	C++ Object Oriented Programming	Prolog Tutorial Exercise due on 10/24 @ 11:59 pm (.pl file)
10	27 Oct	Data Types	6	Intro to Java	C++ Review Exercise due on 10/31 @ 11:59 pm (.cpp file)
11	3 Nov	Expressions & Statements	7	Java OOP	Project Part3 assigned
12	10 Nov	Object Oriented Programming	12	Java List Class and Sorting	
13	17 Nov	Subprogram Implementation	10	Tutorials	Java Tutorial Exercise Due on 11/21 @ 11:59 pm (.tgz file)
14	24 Nov	Control Structures	8	Tutorials	
15	1 Dec	Concurrency	13	Tutorials	Scripting Language Tutorial due on 12/5 @ 11:59 pm (.doc or .pdf file)
16	8 Dec	Exception Handling	14	Tutorial Evaluations	Completed Project due
	15 Dec	Final Exam(2/16 1:00 am -2:50 pm)	Ch 1-16		Final Exam – including Prolog and Java

Policy on Collaboration

You are encouraged to discuss course material with other students. Don't be shy about consulting with anyone, but please understand that you, and only you, bear the responsibility for solving the problems associated with producing a successful project or solving a tutorial assignment. Please read the CS Department policy on plagiarism and keep the following in mind.

All material turned in for credit must be your own work (team assignments are an exception to this). You may discuss ideas and approaches but you should work out all details and write up all solutions on your own. Copying part or all of another student's assignment, with or without the student's knowledge, is prohibited. Similarly, copying old or published solutions is prohibited.

Receive help with care. Avoid working too closely with another student. Otherwise, you can unwittingly become dependent on that student's help and fool yourself into thinking that you understand things better than you really do. Always attempt to do as much as you can on your own. Then, after you do seek help, be sure to work through similar problems on your own. Also, don't forget other sources of programming help such as the your textbook, <http://www.cplusplus.com>, the debugger, and CodeWarrior and Visual C++ documentation.

Give help with care. Don't help too much. When you understand something, you may be tempted to show someone the complete solution. However, if you do this, you will rob them of the learning experience of reaching the solution on their own. Try giving a hint that will help them get "unstuck" Although you are allowed to help other students, you are never under any obligation to do so.

Violations of these restrictions carry severe penalties. Remember that you are ultimately (i.e., during an exam or quiz) responsible for understanding the material.

Policy on Incomplete Grade

It is the policy of the Computer Science Department that a grade of Incomplete (I) shall be assigned only when the instructor concludes that a clearly identifiable portion of course requirements cannot be met within the academic term for unforeseen, but fully justified, reasons; and that there is still a possibility of earning credit.

An incomplete shall NOT be assigned when:

- the request is made before the thirteenth week of instruction
- it is necessary for the student to attend a major portion of the class when it is next offered (i.e., if a student needs to repeat a class, an incomplete should not be given)
- the student is not passing the course with a C- or better at the time of the request
- the student is unable to keep up with course work due to an excessive course load

The condition for removal of the Incomplete shall be entered on the "Request for Incomplete" form and a copy filed in the department office prior to listing an "I" on the Grade Roster. The student must retain the grades for any coursework that was due prior to the incomplete being assigned. The incomplete cannot be removed on the basis of work taken at another institution nor by re-enrolling in the course.

An incomplete must be made up within one calendar year immediately following the end of the term in which it was assigned. This limitation prevails whether or not the student maintains continuous enrollment. Failure to complete the assigned work will result in an incomplete "I" being converted to a "NC" which will affect the grade point average.

Fall 2008 University Calendar

Fall 2008

August 2008						
SU	MO	TU	WE	TH	FR	SA
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31						

- 4 - 8 Fall 2008 Second Registration (by appointment only)
- 19 20 NO REGISTRATION (FA processing)
- 25 Fall 2008 New Student Orientation and Registration.
- 25 Open Registration/Add/Drop
- 25 Last day to drop classes with full refund of fees
- **26 Instruction Begins**
- 26 - Sept. 9 Late Registration/Add/Drop

-
- **1 Labor Day holiday (University closed)**
 - 9 Last day to Add
 - 9 Priority day to submit Contract Courses (Special Studies, Internships, Community Involvement, etc.)
 - 9 Last day to Drop a course(s) (with adjusted fees) (done online)
 - 9 Last day to change from Full Time to Part Time fee status
 - 9 Final Fee Payment deadline. Students dropped for non-payment of fees will not be eligible for reinstatement for Fall 2008.

September 2008						
SU	MO	TU	WE	TH	FR	SA
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30				

- 10 - 22 Drop with a 'W' (done on-line)
- 10 - 22 Petition to Add with \$20 administrative fee per class (adding classes permitted because of serious and compelling reasons only)
- 15 Final deadline to submit Graduation Applications for December 2008
- 15 Graduation Application priority filing date for students planning to graduate May 2009
- 22 Final deadline to submit Contract Courses. Note: class may not be added to your schedule until after the Registration Freeze (October 3)
- 22 Last day to Drop with 'W' (done online)
- 22 Last day to Petition to Add
- 22 Last day to file Grade Appeal from previous semester
- 22 Last day to change Grading Basis (done on-line)
- **23 Census Date**
- 23 - Oct 10 REGISTRATION FREEZE - no processing of adds or drops

- 23 - Nov. 7 Petition to Withdraw from a Class w/ \$20 administrative fee (dropping classes permitted because of serious and compelling reasons)

October 2008						
SU	MO	TU	WE	TH	FR	SA
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

- 3 Registration processing continues

November 2008						
SU	MO	TU	WE	TH	FR	SA
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30						

- 1 ERD due for 1st registration participation
- 1 Last day to drop all classes and receive pro-rated cancellation of fees
- 7 Last day to completely withdraw from the University (no refund)
- 7 Last day to Petition to Withdraw from a Class
- **1 Veteran's Day Observed, campus closed**
- 2 - Dec. 2 Withdrawing from the semester at this point follows same requirements as retroactive withdrawal
- 26 - 28 Thanksgiving holiday for students - No classes the 26th; University closed 27th and 28th

December 2008						
SU	MO	TU	WE	TH	FR	SA
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

- 1 ERD due for 2nd registration participation
- 1 - 5 Spring 2009 Registration (by appointment only.)
- 1 Thesis deadline for December 2008 graduates
- 12 Last day of instruction
- 15 - 19 Finals Week
- **25 – January 1 Winter Holiday (campus closed)**

Tutorial Assignment and Project Submission Instructions

All tutorial assignments and projects will be submitted electronically on the linux platform. Naming standards are important; if you do not follow the standards, I will not receive your submission and you will not receive credit for the submission.

Submitting Tutorial Assignments

1. Make a copy of the program to put in the drop box:
`cp Tutorial2.c yourlastname.c`
2. Copy your new copy to the drop box:
`cp yourlastname.c ~tiawatts/cs460drop/.`

Note the period at the end of each these statements.... it is important!

3. Wait about 2 minutes and then check the web page to verify that your file has reached the drop box:
<http://www.cs.sonoma.edu/~tiawatts/cs460/tut2sub.txt>

Submitting projects

1. Create a directory in your file system called <last_name>P1
eg. wattsP1
2. Copy all of the files you are submitting to your <last_name>P1 directory.
ie. *.c, *.cpp, *.h, makefile
3. Make the parent directory of your <last_name>P1 the current directory.
4. Enter
`umask 033`
5. Enter the follow tar command:
`tar cfvz <last_name>P1.tz <last_name>P1/*`
6. Enter
`cp <last_name>P1.tz ~tiawatts/cs460drop/.`
7. Enter
`umask 077`
8. Wait about 2 minutes and then check the web page to verify that your file has reached the drop box:
<http://www.cs.sonoma.edu/~tiawatts/cs460/proj1sub.txt>

C Review Exercise 1 (Simple Sorting)

Topics: C programming language
Sorting

Goals: Upon successful completion of this tutorial you should be able to:

1. Open and read a C stream file
2. Create and sort a C static array using a simple sort

Related text sections:

C Review Exercise 1 Instructions

This tutorial is designed to help you review your C programming skills. The program you develop for this exercise will be used for future tutorial exercises.

Internal sorting techniques are frequently described as simple or advanced. Simple sorting techniques include the Exchange (bubble), Insertion, and Selection sorts. Advanced techniques include Quicksort and Mergesort.

For this tutorial, you are to write a C program which uses one of the simple sorting techniques to sort a set of integers into ascending order. For C tutorial exercise 2 you will use one of the advanced sorting techniques to sort the same set of integers into descending order.

The sorts you are to implement will depend on the number you draw in class.

1. Exchange and Mergesort
2. Exchange and Quicksort
3. Insertion and Mergesort
4. Insertion and Quicksort
5. Selection and Mergesort
6. Selection and Quicksort

The documentation in your program should clearly indicate and describe the simple sort being implemented. Your program should use 1 or more nested functions.

Input: Your C program should read a text file containing a number of integer values. The file name should be accepted as a command line argument. *The first value in the file is the number of integers to be sorted.* The remaining values are the values to be sorted.

Output: The console output of your C program should be a sorted list of values. Consecutive values in the sorted output list should be separated by a single space.

Sample Input:

```
8 1 4 23 -45 61 28 0 2
```

Sample Output:

```
-45 0 1 4 2 23 61 28
```

C Review Exercise 2 (Advanced Sorting)

Topics: Recursion
C parameter passing

Goals: Upon successful completion of this tutorial you should be able to:

1. Sort a C static array using a recursive advanced sort

Related text sections:

C Review Exercise 2 Instructions

Next week we will start using the functional language Scheme. One of the most powerful characteristics of the functional language paradigm is the ability to implement algorithms that employ recursion. For this tutorial, as a review of recursion, you are to add the implementation of a recursive, descending sort to the C program you created for C tutorial 1.

The recursive sort you are to implement will depend on the number you drew in class for tutorial 1.

1. Exchange and Mergesort
2. Exchange and Quicksort
3. Insertion and Mergesort
4. Insertion and Quicksort
5. Selection and Mergesort
6. Selection and Quicksort

The documentation in your program should clearly indicate and describe the recursive sort being implemented.

Input: Your C program should read a text file containing a number of integer values. The file name should be accepted as a command line argument. The first value in the file is the number of integers to be sorted. The remaining values are the values to be sorted.

Output: The console output of your C program should be a sorted list of values. The values should first be sorted into ascending order using your simple sort and then into descending order using your advanced sort. Each sort should start with a copy of the original list read from the file. Consecutive values in each sorted output list should be separated by a single space. The two lists of sorted values should be separated by one blank line.

Sample Input:

```
8 1 4 23 -45 61 28 0 2
```

Sample Output:

```
-45 0 1 2 4 23 61 28
```

```
61 28 23 4 2 1 0 -45
```

To turn in: A listing of your well formatted and documented C program. Copy your program to `~tiawatts/cs460drop` as `your_last_name.c` and ask me to test it.

Scheme Tutorial Exercise 1 (Introduction to Scheme)

Topics: Beginning Scheme

Goals: Upon successful completion of this tutorial you should be able to:

1. Define basic Scheme terms
2. Manipulate a list using Scheme primitives
3. Define Scheme functions

Related text sections:

Scheme Tutorial Exercise 1 Instructions

This tutorial is designed to give you an introduction to the functional language Scheme.

To start the Scheme interpreter on the departmental linux server enter `guile` at the linux prompt. While running the guile Scheme interpreter commands will be entered at the `guile>` prompt. Enter `ctrl-d` or `(exit)` to exit from the interpreter.

Enter each of the following commands at the scheme prompt and record the result.

```
(car '(a b c))
```

```
(cdr '(a b c))
```

```
(cons 'a '(b c))
```

Based on the above responses, describe the function of each of the scheme functions represented:

car:

cdr:

cons:

Predict the output of the following scheme command:

```
(cons (car '(a b c))  
      (cdr '(d e f)))
```

Enter this command to determine whether your prediction was correct.

As you can see, Scheme expressions may span more than one line. The Scheme system knows when it has an entire expression by matching double quotes and parentheses.

Enter the following Scheme procedure definition:

```
(define square  
  (lambda (n)  
    (* n n)))
```

The procedure `square` computes the square n^2 of any number n . `Define` establishes variable bindings, `lambda` creates procedures, and `*` names the multiplication procedure. Note the form of these expressions. All structured forms are enclosed in parentheses and written in prefix notation, i.e., the operator precedes the arguments. As you can see, this is true even for simple arithmetic operations such as `*`.

Try using `square` by entering each of the following commands and recording its result:

```
(square 5)
```

```
(square -200)
```

```
(square 0.5)
```

(square -1/2)

Note: Scheme systems that do not support exact ratios internally may print 0.25 for (square -1/2).

Even though the next definition is short, please enter it using a file. Exit from the scheme interpreter by entering ctrl-d. Call the file "reciprocal.ss."

```
(define reciprocal
  (lambda (n)
    (if (= n 0)
        "oops!"
        (/ 1 n))))
```

This procedure, reciprocal, computes the quantity $1/n$ for any number n . For $n = 0$, reciprocal returns the string "oops!". Return to Scheme and load your file with the procedure load.

```
(load "reciprocal.ss")
```

Try using the reciprocal procedure you have just defined by entering and recording the result of each of the following commands.

```
(reciprocal 10)
```

```
(reciprocal 1/10)
```

```
(reciprocal 0)
```

```
(reciprocal (reciprocal 1/10))
```

Write and execute two commands that use both the reciprocal and square procedures (Note: you will need to redefine the square procedure to test your commands.) Be creative!

1. Command:

Result:

2. Command:

Result:

Scheme arithmetic expressions are expressed using the standard operators (+ - * /) and prefix notation. Enter and record the result of each of the following Scheme commands:

```
(+ 4 5)
```

```
(* 1.5 2.3)
```

```
(- (* 4 4/5) (/ 4 5))
```

Write and execute Scheme commands to perform each of the following calculations:

$$1.2 \times (2 - 1/3) + -8.7$$

$$(2/3 + 4/9)/(5/1 - 4/3)$$

$$1 + 1/(2 + 1/(1 + 1/2))$$

$$1 \times -2 \times 3 \times -4 \times 5 \times -6 \times 7$$

Write and execute two commands that use simple arithmetic operations and the reciprocal and square procedures. Be creative!

1. Command:

Result:

2. Command:

Result:

Using your book and resources found on the web, define each of the following terms used by the Scheme programming language:

atom:

primitive:

lambda expression:

free variable:

top level definition:

predicate:

type predicate:

lexical scoping:

Scheme Tutorial Exercise 2 (Scheme Lists)

Topics: Advanced Scheme functions

Goals: Upon successful completion of this tutorial you should be able to:

1. Design and implement Scheme functions to manipulate the items in a list

Related text sections:

Scheme Tutorial Exercise 2 Instructions

1. Describe the function and use of each of the following Scheme primitive functions:
 - a. list?
 - b. number?
 - c. null?
 - d. caddr
 - e. if
 - f. let
2. Write and test a Scheme function called `insert-first` that will insert a value into a list as the first element of the list. The formal arguments should be a list and the value to be inserted into the list.
3. Write and test a Scheme function called `remove-first` that will remove the first element of a list. The formal argument should be a list.
4. What happens when you enter `(insert-first 1 2)`? What happens when you enter `(remove-first 1)`? Modify the functions you wrote in steps 1 and 2 to test their input values to prevent these problems. If the input values are incorrect, your functions should issue an informative error message.
5. What happens if you enter `(remove-first '())`? Modify your `remove-first` function to test its input value to prevent this problem. If the input list is empty, your function should issue an informative error message.
6.
 - a. Write and test a recursive Scheme function called `list-copy` that will create a copy of a list. The formal argument should be a list.
 - b. Write and test a recursive Scheme function called `odd-copy` that will create a copy of the elements in the odd numbered positions in a list starting with the first element in the list. The formal argument should be a list.
 - c. Write and test a recursive Scheme function called `even-copy` that will create a copy of the elements in the even numbered positions in a list starting with the second element in the list. The formal argument should be a list.
7. Write and test a Scheme function called `insert-last` that will insert a value into a list as the last element of the list. The formal arguments should be a list and the value to be inserted into the list.
8. Write and test a Scheme function called `remove-last` that will remove the last element of a list. The formal argument should be a list.
9. Write and test a Scheme function called `list-reverse` that will reverse the elements of a list. The formal argument should be a list.

Scheme Tutorial Exercise 3 (Scheme Sorting)

Topics: List sorting using Scheme

Goals: Upon successful completion of this tutorial you should be able to:

1. Design and implement Scheme functions to recursively sort the items in a list

Related text sections:

Scheme Tutorial Exercise 3 Instructions

1. Execute the list reversing procedure you wrote for Tutorial 2 using '(a (b c) d) as input. Does it return (d (c b) a)? Or does it return (d (b c) a)? Write a new procedure called all-reverse that recursively reverses nested lists. Execute your new procedure using '(a (b (c d)) (e f)) as input. It should return ((f e) ((d c) b) a).
2. Each of the recursive sorts we have discussed and implemented are of the form:

```
recursively-sort (list L)
{
  if (size of L > 1)
  {
    split L into 2 parts: L1 and L2
    recursively-sort L1
    recursively-sort L2
    combine sorted lists L1 and L2 into a sorted list L
  }
}
```

2. a. Write a scheme procedure called quicksort that uses Quick Sort to sort its single list argument into **ascending** order. You will probably need to create a helper function (or functions) to split the list into 2 parts – those elements < than the pivot and those >= to the pivot. You may also need an append procedure.

OR

2. b. Write a scheme procedure called mergesort that uses Merge Sort to sort its single list argument into **descending** order. You will probably need to create a helper function (or functions) to split the list into 2 parts. You may also need a merge procedure.
3. Run each of your sorts using the list '(1 5 3 6 8 92 -1 0 4 5 3). Run each of your sorts using the list '(a b d e c t s). How do your sorts work with these input lists? Modify your sorts so that they sort only lists of numeric values.
4. Place the well documented procedures you created for Scheme tutorials 2 and 3 in a file called *your-last-name.ss* and drop the file into the cs460drop folder.

Prolog Tutorial Exercise 1 (Intro to Prolog)

Topics: Logic programming and Prolog terms
Prolog rules, facts, and queries

Goals: Upon successful completion of this tutorial you should be able to:

1. Write a simple Prolog rule
2. Create a base of facts for a prolog session
3. Write a simple Prolog query

Related text sections:

Prolog Tutorial Exercise 1 Instructions

For this tutorial you will be starting your journey into the world of Prolog! This tutorial is divided into 3 parts. The first part of this tutorial consists of a set of questions regarding the language Prolog. You may use web and/or printed resources to answer these questions. The second part of this tutorial is a directed Prolog exercise using gprolog on the linux platform. In the third part of this tutorial you will be creating a set of Prolog facts and queries.

Tutorial 1 Part 1.

1. To what programming paradigm family does the language Prolog belong?
2. For what types of applications is Prolog an appropriate programming language?
3. Describe each of the following terms with respect to the Prolog programming language:
 - a. world
 - b. fact
 - c. rule
 - d. base
 - e. predicates
 - f. arguments
 - g. arity
 - h. list
 - i. instantiation
 - j. unification
4. Describe the Prolog syntax rules regarding each of the following:
 - a. selecting variable names
 - b. defining constant (literal) values
 - c. use of periods (.)
 - d. use of semicolons (;)
 - e. use of commas (,)
 - f. formulation of facts
 - g. formulation of queries
 - h. formulation of rules
 - i. assignment statements
 - j. conditional expressions
5. How do you create a Prolog source file? How do you document a Prolog source file? How do you load the file when using the Prolog interpreter?

Tutorial 1 Part 2.

1. Copy the files forecast6.pl and photoshoot6.pl from ~tiawatts/cs460pickup to your account.
2. Start the Gnu Prolog interpreter by entering **gprolog** at the linux prompt.
3. Load the file forecast6.pl by entering [**forecast.pl**]. at the Prolog interpreter prompt.

4. List the contents of the file by entering **listing.** at the Prolog interpreter prompt.
5. Query the (data)base to determine if Friday will be sunny by entering **weather(friday,sunny).** at the Prolog interpreter prompt.
6. Query the (data)base to determine if Saturday will be rainy.
7. Query the (data)base for Monday's weather forecast by entering **weather(monday,What).** at the Prolog interpreter prompt.
8. Create and enter a query for today's weather forecast.
9. Query the (data)base for a rainy day by entering **weather(Day,rainy).** at the Prolog interpreter prompt.
10. Create and enter a query for a snowy day.
11. Create and enter a query for fair days; use the semicolon operator to continue the query.
12. Exit from the Prolog interpreter by entering **halt.**
13. Restart the Gnu Prolog interpreter and load and compile the file photshoot6.pl.
14. The file 'photoshoot.pl' contains the line: **sky(blue, A) :- weather(A, fair).** What does this statement mean?
15. Execute the query: **sky (blue, friday).** What is the result of this query?
16. Write a query to determine the color Friday's sky.
17. Write a query to determine the color of today's sky.
18. The file 'photoshoot.pl' contains the line: **goodpictures(A) :- sky(blue, A), weekend(A).** What does this statement mean?
19. Execute the query: **goodpictures(monday).** What is the result of this query? How do you think Prolog arrives at this conclusion?
20. Turn on the Prolog trace option (by entering trace.) and reenter the above query. Press the enter key to move from one Prolog operation to the next.
21. Enter and trace the query: **goodpictures(D).** What is the result of this query?
22. Exit from Prolog.

Tutorial 1 Part 3.

1. For a topic of your choice, create a set of 10 facts and enter them into a prolog data base file.
2. Create 2 simple rules and and one compound (conjunction or disjunction) rule for your data base.

3. Create a list of 10 queries and their results for your program.
4. Use gprolog to compile your data base and test your queries.

Prolog Tutorial Exercise 2 (Prolog “Functions” and Lists)

Topics: Prolog compound rules
Prolog List structures

Goals: Upon successful completion of this tutorial you should be able to:

1. Write a compound prolog rule
2. Manipulate the items in a prolog list

Related text sections:

Prolog Tutorial Exercise 2 Instructions

Tutorial 2 Part 1.

1. Copy the files factorial.pl and list.pl from ~tiawatts/cs460pickup to your account.
2. Start the Gnu Prolog interpreter and load the file factorial.pl.

```
factorial(0,1).  
  
factorial(A,B) :-  
    A > 0,  
    C is A-1,  
    factorial(C,D),  
    B is A*D.
```

1. Enter the query **factorial(5,What)**. Explain this result:
2. Enter the query **factorial(0,Value)**. Explain this result:
3. Analyze the factorial function; you should consider the following questions:
 - a) What is the role of the statement `factorial(0,1).` in the definition of factorial?
 - b) What is the role of the statement `A > 0` in the definition of factorial?
 - c) What is the role of the statement `factorial(C,D)` in the definition of factorial?
 - d) Explain the pattern of variable uses in the last four lines.
4. Exit from the Prolog interpreter.

Tutorial 2 Part 2.

1. What is a Prolog “list”?
2. What is the syntax used to create a Prolog list?
3. How are the terms “head” and “tail” defined with respect to a Prolog list?
4. How is membership in a list determined?
5. How is an empty list represented?

Tutorial 2 Part 3.

1. Start the Gnu Prolog interpreter and load the file list.pl. Use the following activities to analyze the statements in the file list.pl.

2. Use the [user] command to create 2 list facts, list1 and list2.


```
[user].
list1([a,b,c,d,e]).
list2([x,y,z]).
Ctrl-D
```
3. Enter the following query to determine if a is the first element in list1:


```
list1(L),is_first(a,L).
```
4. Enter the following query to determine the identity of the first element in list2:


```
list2(L),is_first(X,L).
```
5. Create similar queries to test is_last, is_in, how_long, and place.
6. Add comments to the file list.pl to explain how each of these rules works. (You may wish to use the trace command to view these rules in action).
7. Enter the following command to push the item m onto the stack represented by list1.


```
list1(List),insert_first(m,List,Newlist).
```
8. Enter the following command to pop the first item from the stack represented by list2.


```
list2(List),remove_first(List,Newlist).
```
9. Enter the following command to concatenate 2 lists.


```
concat([a,b,c],[d,e,f],L).
```
10. Create a query that uses concat to determine if one list is the prefix of another. Create a query that uses concat to determine if one list is the suffix of another.
1. Add comments to the file list.pl to explain how each of these rules works. (You may wish to use the trace command to view these rules in action).

Tutorial 2 Part 4.

1. Add a rule called insert_last to this set of rules. insert_last should have 3 arguments: an element, a list, and a resulting list. The element should be appended to the end of the original list to create the resulting list.
2. Add a rule called remove_last to this set of rules. remove_last should have 2 arguments: an original list and a modified list. The last element of the original list should be removed to create the modified list.
3. Add a rule called switch. switch should have 2 arguments: an original list and a modified list. The elements in the original list should be reversed to create the modified list.

Prolog Tutorial Exercise 3 (Prolog Sorting)

Topics: Sorting in Prolog

Goals: Upon successful completion of this tutorial you should be able to:

1. Use prolog rules of unification to create a sorted list

Related text sections:

Prolog Tutorial Exercise 3 Instructions

1. The following rules provide the framework for a Prolog implementation of quicksort. Using the list rules you created for tutorial2, write rules for quick_split/4. Test your quick_split and quicksort rules. Place these rules in a file called sort.pl.

```
quicksort([], []).
quicksort([Element],[Element]).
quicksort([Head|Tail],SortedList) :-
    quick_split(Head,Tail,L,H),
    quicksort(L,SL),
    quicksort(H,SH),

    concat(SL,[Head|SH],SortedList).
```

OR/AND

- The following rules provide the framework for a Prolog implementation of mergesort. Using the list rules you created for tutorial2, write rules for merge_split/3 and merge/3. Test your merge_split, merge and mergesort rules. Place these rules in a file called sort.pl.

```
mergesort([], []).
mergesort([Element],[Element]).
mergesort(List,SortedList) :-
    merge_split(List,L1,L2),
    mergesort(L1,SL1),
    mergesort(L2,SL2),
    merge(SL1,SL2,SortedList).
```

2. Test your sort(s) using the rules in cs460lpickup/tutorial3.pl and the following gprolog script:

```
$ cat tutorial3.pl sort.pl > temp.pl
$ gprolog
GNU Prolog 1.2.8
By Daniel Diaz
Copyright (C) 1999-2001 Daniel Diaz
| ?- [temp].
compiling temp.pl for byte code...
temp.pl compiled, 96 lines read - 2978 bytes written, 39 ms
(10 ms) yes
| ?- setup.
yes
| ?- run1(list0,list1),ilist(list1,L).
L = [-3,-2,-1,1,1,2,2,3,3] ?
yes
| ?- run2(list0,list2),ilist(list2,L).
L = [3,3,2,2,1,1,-1,-2,-3] ?
yes
| ?- halt.
$
```

3. Place your well documented rules for tutorials 2 and 3 in a file called *your-last-name.pl* and drop the file into the cs460drop folder.

C++ Review Exercise 1 (C++ OOP and Sorting)

Topics: C++ classes and objects

Goals: Upon successful completion of this tutorial you should be able to:

1. Create and sort your own list class using an array
2. Create and sort your own list class using 1 linked list

Related text sections:

C++ Review Exercise 1 Instructions

Next week we will start working with the object oriented language, Java.

For this tutorial, as a review of C++ OOP, you are to implement a template container class called List. List should use dynamic memory allocation.

Your class should contain the following member or friend methods:

1. A default constructor: List(). The constructor should create an empty list.
2. A destructor: ~List(). The destructor should release all dynamically allocated memory space.
3. A copy constructor: List (const List & other). This constructor should create a new list that is a deep copy of the other list passed as a parameter.
4. An assignment operator: List & operator = List (const List & other). This operator should release space allocated to the current list object the create a list that is a deep copy of the other list passed as a parameter.
5. A sorting function: void Sort (SortType ad). SortType should be defined as an enumerated type containing the values ASCENDING and DESCENDING. When the passed value is ASCENDING, the values in the list should be sorted into ascending order using the simple sort you selected for Tutorial 2 (Exchange, Insertion, or Selection). When the passed value is DESCENDING, the values in the list should be sorted into descending order using the advanced sort you selected for Tutorial 2 (Merge or Quick).
6. An input operator: >>. The input operator should read from a character stream. Values should be placed in the list as they are read. A value of a non-compatible type or the end-of-file marker should terminate the input process. White space should not be included in the list.
7. An output operator: <<. The output operator should write the values in the list to a character stream. Consecutive output values should be separated by a single space character.

A driver program (called main.cpp) for testing your List class can be found in ~tiawatts /cs460pickup.

Your completed template file should be placed in ~tiawatts/cs460drop. The file should be called *your-last-name.cpp*.

Java Tutorial Exercise 1 (Introduction to Java)

Topics: Java Libraries – Swing
Event driven programming

Goals: Upon successful completion of this tutorial you should be able to:

1. Write a simple event driven Java program using the Swing libraries

Related text sections:

Java Tutorial Exercise 1 Instructions

1. Define the following java terms and concepts
 - a. class
 - b. object
 - c. primitive
2. The remainder of this tutorial exercise should be executed on the local linux or DOS platforms. You can store your directories and files in the local student directory (in which case you will need to delete them at the end of the tutorial session) or on a flash drive. Completed exercises should be uploaded to the server for submission using scp or ftp. Create a new directory called Tutorial1;
3. Enter the following Java code in a file called Tutorial1a.java in your Tutorial1 directory.

```
// file: Tutorial1a.java

import java.awt.*;
import javax.swing.*;

public class Tutorial1a extends JPanel
{
    int hwX, hwY;
    String helloWorld = null;

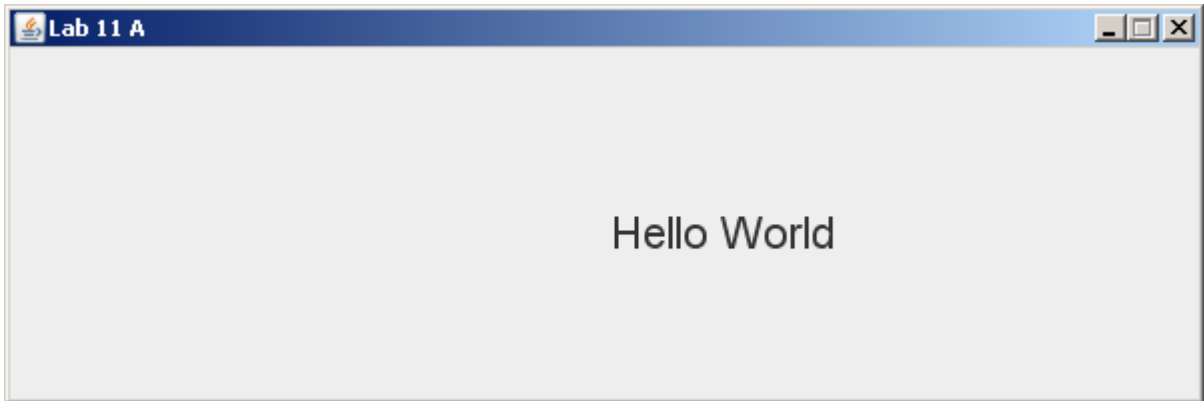
    public Tutorial1a ()
    {
        hwX = 300;
        hwY = 300;
        helloWorld = "Hello World";
        repaint();
    }

    public void paintComponent (Graphics g)
    {
        super.paintComponent (g);
        Graphics2D g2 = (Graphics2D) g;
        g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
            RenderingHints.VALUE_ANTIALIAS_ON);
        g2.setFont (new Font ("Arial", Font.PLAIN, 22));
        g2.drawString (helloWorld, hwX, hwY);
    }

    public static void main (String[] args)
    {
        JFrame frame = new JFrame ("Tutorial 1 A");
        Tutorial1a Tutorial1a = new Tutorial1a ();
        frame.getContentPane().add (Tutorial1a);
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
        frame.setSize (600,600);
        frame.setVisible (true);
        frame.setResizable (false);
        frame.setLocation (100, 100);
    }
}
```

4. Compile and execute your program:
javac Tutorial1a.java
java Tutorial1a

You should see:



You have now written your first Java program (for this course)!

5. Copy Tutorial1a.java to Tutorial1b.java. Modify the program as indicated:

```
// file: Tutorial1b.java

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class Tutorial1b extends JPanel implements ActionListener,
    MouseMotionListener
{
    JButton theButton;
    int hwX, hwY;
    String helloWorld = null;
    int red, green, blue;

    public Tutorial1b ()
    {
        hwX = 300;
        hwY = 300;
        helloWorld = "Hello World";
        red = green = blue = 30;
        theButton = new JButton ("Change Color");
        add (theButton);
        theButton.addActionListener (this);
        addMouseMotionListener(this);
        repaint();
    }

    public void mouseDragged(MouseEvent e)
    {
        hwX = e.getX();
        hwY = e.getY();
        repaint();
    }
}
```

```

public void mouseMoved(MouseEvent e) {}

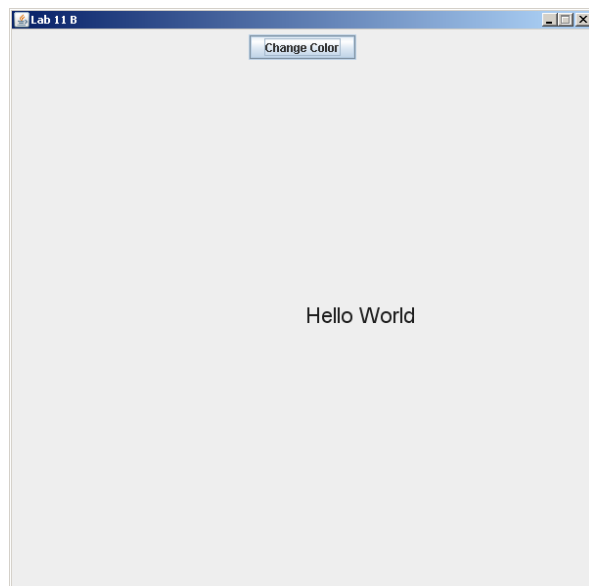
public void paintComponent (Graphics g)
{
    super.paintComponent (g);
    Graphics2D g2 = (Graphics2D) g;
    g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
        RenderingHints.VALUE_ANTIALIAS_ON);
    g2.setPaint (new Color (red, green, blue));
    g2.setFont (new Font ("Arial", Font.PLAIN, 22));
    g2.drawString (helloWorld, hwX, hwY);
}

public void actionPerformed (ActionEvent e)
{
    if (e.getSource() == theButton)
    {
        red = (red * 29) % 254 + 2;
        green = (green * 31) % 254 + 2;
        blue = (blue * 53) % 254 + 2;
        repaint ();
    }
}

public static void main (String[] args)
{
    JFrame frame = new JFrame ("Tutorial 1 B");
    Tutorial1b Tutorial1b = new Tutorial1b ();
    frame.getContentPane().add (Tutorial1b);
    frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
    frame.setSize (600,600);
    frame.setVisible (true);
    frame.setResizable (false);
    frame.setLocation (200, 200);
}
}

```

6. Compile and execute the program; you should see:
7. Use the button to change the color of the text and use the mouse to drag the text around the screen.



8. Copy Tutorial1b.java to Tutorial1c.java. Make the changes necessary to compile and execute the class Tutorial1c.

9. Add a Mouse Listener to Tutorial1c:

```
public class Tutorial1b extends JPanel implements ActionListener,  
MouseMotionListener, MouseListener
```

You will need to include the following MouseListener method stubs:

```
public void mousePressed(MouseEvent e) {}  
public void mouseReleased(MouseEvent e) {}  
public void mouseEntered(MouseEvent e) {}  
public void mouseExited(MouseEvent e) {}  
public void mouseClicked(MouseEvent e) {}
```

10. Modify the mouseExited and mouseEntered methods to prevent the user from dragging the text outside the bounds of the window frame.

Java Tutorial Exercise 2 (Java Lists and Sorting)

Topics: Java Lists

Goals: Upon successful completion of this tutorial you should be able to:

1. Create an array based list in a java program
2. Randomly populate a list of integers
3. Sort the list of integers using your simple and advanced sorts

Related text sections:

Java Tutorial Exercise 2 Instructions

Create a new subdirectory called Tutorial2 and enter the following java program as RList.java

```
// file: RList.java

import java.lang.*;
import java.util.Random;

public class RList
{
    int size;
    int list [];
    static int ASCENDING = 1;
    static int DESCENDING = 2;

    public RList ( )
    {
    }

    public RList (int S)
    {
        size = S;
        list = new int [size];
        for (int i = 0; i < size; i++)
            list[i] = i * 3 % (2 * i + 1);
    }

    public RList (RList other)
    {
        size = other.size;
        list = new int [size];
        for (int i = 0; i < size; i++)
            list[i] = other.list[i];
    }

    public String toString ( )
    {
        String string = "";
        for (int i = 0; i < size; i++)
            string += list[i] + " ";
        return string;
    }

    public void Sort (int AD)
    {
    }

    public static void main (String[] args)
    {
        RList rlist1 = new RList (10);
        RList rlist2 = new RList (rlist1);
        System.out.println ("\nRandom List 1");
        System.out.println (rlist1);
        rlist1.Sort (RList.ASCENDING);
        System.out.println (rlist1);
    }
}
```

```

        System.out.println ("\nRandom List 2");
        System.out.println (rlist2);
        rlist2.Sort (RList.DESCEDING);
        System.out.println (rlist2);
    }
}

```

1. Compile and execute the program:

```

javac RList.java
java RList

```

On the system console you should see:

```

Random List 1
0 1 1 4 7 10 0 1 2 3
0 1 1 4 7 10 0 1 2 3

Random List 2
0 1 1 4 7 10 0 1 2 3
0 1 1 4 7 10 0 1 2 3

```

Note that the lists are not sorted.....

2. Modify the Sort method of the RList class to sort the list into ascending order using your simple sort if the input parameter is ASCENDING and into descending order using your advanced sort if the input parameter is DESCENDING.
3. This program uses an internal main function to test the RList class. The RList class can also be used as part of another program. Enter, compile and run the following class called Tutorial2a.

```

// file: Tutorial2a.java

import java.lang.*;

public class Tutorial2a
{
    RList rlist1;
    RList rlist2;

    public Tutorial2a ()
    {
        rlist1 = new RList (10);
        rlist2 = new RList (rlist1);
        System.out.println ("\nRandom List 1");
        System.out.println (rlist1.toString());
        rlist1.Sort (RList.ASCENDING);
        System.out.println (rlist1.toString());
        System.out.println ("\nRandom List 2");
        System.out.println (rlist2.toString());
        rlist2.Sort (RList.DESCEDING);
        System.out.println (rlist2.toString());
    }

    public static void main (String[] args)
    {
        Tutorial2a tutorial2a = new Tutorial2a ();
    }
}

```

4. Modify the RList class to include a random number generator. Information about the java Random class can be found at:

<http://java.sun.com/j2se/1.4.2/docs/api/java/util/Random.html>

Modify the constructor with integer input parameter `S` to generate a list populated with `S` randomly selected integers in the range $-100 \leq \text{list}[i] \leq 100$.

Modify the default constructor to randomly select a value for `size` ($5 \leq \text{size} \leq 50$) and to generate a list populated with `size` randomly selected integers in the range $-100 \leq \text{list}[i] \leq 100$.

Java Tutorial Exercise 3 (Java Illustration)

Topics: Java 2D graphics

Goals: Upon successful completion of this tutorial you should be able to:

1. Use Swing and Java graphics to illustrate the actions of your simple sorting algorithm
2. Use Swing and Java graphics to illustrate the actions of your advanced sorting algorithm

Related text sections:

Java Tutorial Exercise 3 Instructions

Create a new subdirectory called Tutorial3. Copy RList.java from Tutorial2 to Tutorial3. Add the following Paint method to RList.

```
public void Paint (Graphics2D g2, int ulX, int ulY, int lrX, int lrY,
                  Color minColor, Color maxColor)
{
    if (size < 1)
        return;
    int min = 0;
    int max = 0;
    for (int i = 1; i < size; i++)
    {
        if (list[i] < min) min = list[i];
        if (list[i] > max) max = list[i];
    }
    int deltaX = (lrX - ulX) / (2 * size + 1);
    int range = max - min + 1;
    int deltaY = (lrY - ulY) / range;
    int deltaR = (maxColor.getRed() - minColor.getRed()) / range;
    int deltaG = (maxColor.getGreen() - minColor.getGreen()) / range;
    int deltaB = (maxColor.getBlue() - minColor.getBlue()) / range;
    for (int i = 0; i < size; i++)
    {
        int left = ulX + (2*i+1) * deltaX;
        int right = left + deltaX;
        int top = lrY - list[i] * deltaY;
        int bottom = lrY;
        Color c = new Color (list[i] * deltaG, list[i] * deltaR,
                             list[i] * deltaB);
        g2.setPaint (c);
        g2.fillRect (left, top, right-left, bottom-top);
    }
}
```

This function is designed to draw vertical bars representing the values in the list using the rectangle and colors passed to it. The parameters to this function are, in order, the x and y coordinates of the upper left corner of the rectangle in which the vertical bars will be drawn; the x and y coordinates of the lower right corner of the rectangle; the minimum and maximum colors. The width of the rectangular area is divided so that 'size' bars can be drawn along a horizontal axis. The height of the rectangular area is divided into units representing the range of the magnitudes of the values in the list. The color range is divided into units representing the range of the magnitudes of the values in the list.

Currently this function does not handle negative integer values in the list.

1. Enter the following java class:

```
// file:Tutorial3a.java
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class Tutorial3a extends JPanel implements ActionListener
{
```

```

RList rlist;
JButton theButton;
public Tutorial3a ()
{
    rlist = new RList ();
    System.out.println (rlist);
    theButton = new JButton ("Sort List");
    add (theButton);
    theButton.addActionListener (this);
    repaint();
}

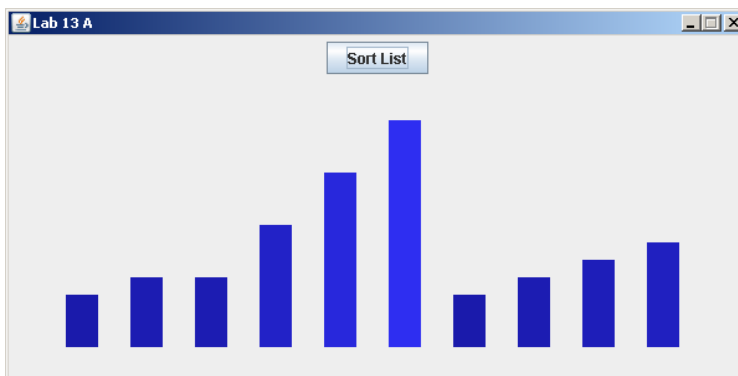
public void paintComponent (Graphics g)
{
    super.paintComponent (g);
    Graphics2D g2 = (Graphics2D) g;
    g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
        RenderingHints.VALUE_ANTIALIAS_ON);
    rlist.Paint (g2, 20, 50, 580, 250, new Color (20, 20, 150),
        new Color (50, 50, 250));
}

public void actionPerformed (ActionEvent e)
{
    if (e.getSource() == theButton)
    {
        rlist.Sort (RList.ASCENDING);
        repaint();
    }
}

public static void main (String[] args)
{
    JFrame frame = new JFrame ("Tutorial 3 A");
    Tutorial3a Tutorial3a = new Tutorial3a ();
    frame.getContentPane().add (Tutorial3a);
    frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
    frame.setSize (600,300);
    frame.setVisible (true);
    frame.setResizable (false);
    frame.setLocation (200, 200);
}
}

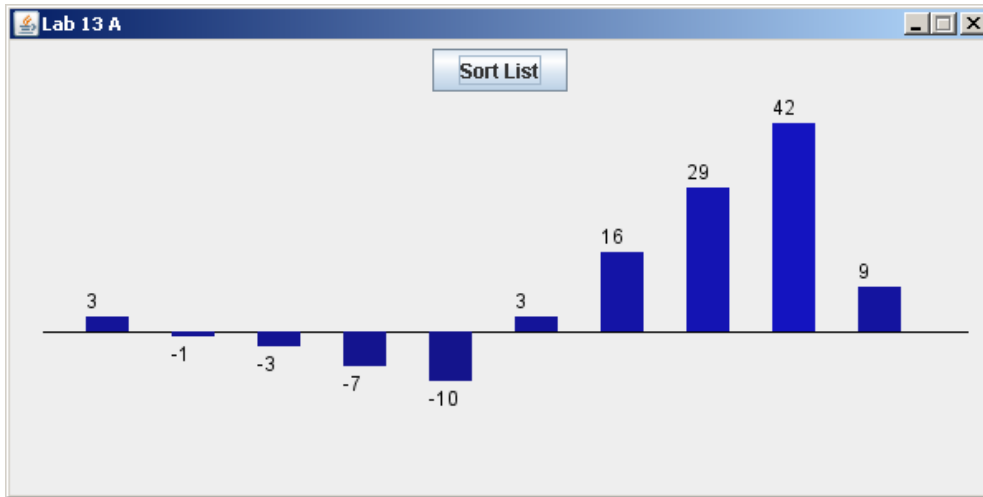
```

2. Modify your constructor so generates numbers between 0 and 100. Compile program. You should see something

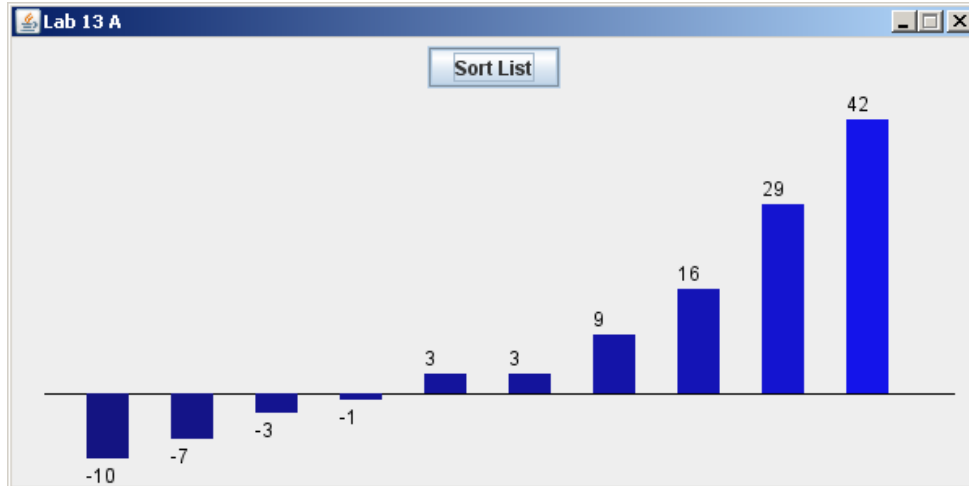


second list that it only positive between 0 and 100 and test the program. You should see something like:

3. Modify the RList Paint function as follows:
 - a. Include a horizontal axis
 - b. Label each vertical bar with its axis
 - c. Allow negative values – these values should appear as vertical bars below the horizontal axis.
- Return your second constructor to its original state (generating numbers between -100 and 100) and test your program. Your display *might* look like:



Test the simple sort function in your RList class. After the Sort List button is pressed, the list should be displayed in fully sorted order. Your display *might* look like:



4. Modify the program so that each press of the “Sort List” button causes 1 pass of your simple sort to be executed. Multiple presses of the button will illustrate the modifications made to the list that eventually result in a fully sorted list. You will most likely need to add a “SortPass” method to your RList class.

Your completed RList.java and Tutorial3a.java files should be placed in ~tiawatts/cs460drop. The files should be called *your-last-name.java* and *your-last-nameT3.java* respectively.

Chapter 1 Review Questions

1. Why do we study the concepts of programming languages?
2. What is a programming domain? Why are programming domains significant? What programming domains are presented in your book? Can you describe a programming domain not included in those listed in your book?
3. When evaluating a programming language, what aspects contribute to its readability?
4. What is orthogonality? How does orthogonality differ from functionality?
5. When evaluating a programming language, what aspects contribute to its writability?
6. What is abstraction? What constructs are provided by C++ to implement data abstraction?
7. When evaluating a programming language, what aspects contribute to its reliability?
8. How does the architecture of a computer contribute to the design of a programming language?
9. What are the four categories of programming languages?
10. What are the five main tasks required of a compiler?
11. What are the differences between compilation and interpretation? When would you choose to use each of these techniques?
12. What are the typical components of a programming environment?

Chapter 2 Review Questions

1. List three features of the language Planalkül that are found in modern programming languages.
2. Early versions of FORTRAN included an arithmetic if statement. Rewrite the following arithmetic if statement as a logical if statement:

```
IF A+B*C 10, 20, 30
```
3. List 3 ways in which FORTRAN 90 differs from FORTRAN 77.
4. For what programming domain was the language LISP designed? Who were the authors of LISP?
5. What 2 dialects of LISP are widely used today?
6. Why has ALGOL had such a great influence on the development of modern programming languages? What are the successful features of ALGOL? What are ALGOL's failures?
7. What features of COBOL have made it a long lived programming language?
8. Who authored the COBOL programming language?
9. What 2 descendents of BASIC are widely used today?
10. Which programming language represented the first attempt to design a programming language suitable to a wide range of programming domains?
11. What did the languages APL and SNOBOL contribute to the programming language landscape?
12. What language first introduced the concept of data abstraction?
13. What was the language Pascal's contribution to the programming language landscape?
14. Who are the authors of the C programming language?
15. List 3 important aspects of the design process of Ada.
16. What language first introduced the concepts of object oriented programming?
17. Who began the transformation of the language C to C++?
18. List 2 contributions of the language Java to the programming language landscape.
19. Why is Java considered to be a "safer" language than C++?
20. List 5 features of a successful programming language.

Chapter 3 Review Questions

1. List the phases on a modern compiler.
2. Differentiate between the front-end and back-end phases of a compiler. Why do we make this distinction?
3. Differentiate between the concepts of “syntax” and “semantics” in a programming language.
4. What is the relationship between a lexeme and a token?
5. What is the function of a parser?
6. What 3 sets are used to define a context free grammar?
7. Given the CFG described below, create a parse tree for the statement:

A := B + C * D

Is this the only possible parse tree for this statement using this grammar?

Show a leftmost derivation for this statement.

CFG:

```
Terminals = { :=, +, *, A, B, C, D}
Non-terminals = { <assign>, <id>, <expr> }
Productions = { <assign> → <id> := <expr>
               <id> → A | B | C | D
               <expr> → <id> + <expr>
                       | <id> * <expr>
                       | <id>
               }
```

8. What are the characteristics of an unambiguous grammar?
9. What is the purpose of an attribute grammar? Of axiomatic semantics? Of denotational semantics?

Chapter 4 Review Questions

1. What is the purpose of a lexical analyzer?
2. What is a regular expression? What is a state diagram (aka DFA)? How are regular expressions and state diagrams related to lexical analysis?
3. What is the purpose of parsing? How is parsing related to the CFG for a language?
4. What is a recursive descent parser?
5. What is a LL(1) grammar?
6. Differentiate between top-down and bottom-up parsing.
7. What types of grammars work best for creating top-down parsers? For creating bottom-up parsers?

Chapter 5 Review Questions

1. What design issues should be considered when choosing identifier naming rules for a programming language?
2. What is a keyword? A reserved word? How do keywords and reserved words differ?
3. How does the used of special words in FORTRAN differ from the use of special words in C or C++?
4. What is a variable? A type?
5. What is binding?
6. What is an alias? How is aliasing used in C/C++?
7. What is a static variable?
8. What is type inference?
9. When are variables typically stored on the program stack?
10. When are variables typically stored in the memory heap?
11. Given the Pascal declarations:

```
var
```

```
    A, B : array [1..10] of integer;
```

```
    C, D : array [1..10] of integer;
```

Why is the statements $A := B$ permissible but the statement $A := C$ is not?

Is the statement $A[\text{index}] := C[\text{index}]$ permissible? Why or why not?

12. How does strong typing differ from weak typing?
13. Differentiate between the following scoping styles: global, local, control structure.
14. Given the following procedure written in “Pascal-like” syntax:

```
Procedure P1;
```

```
    var x : integer;
```

```
    Procedure P2;
```

```
    begin
```

```
        Write (x);
```

```
    end;
```

```
    Procedure P3;
```

```
    var x : integer;
```

```
    begin
```

```
        x := 5;
```

```
        P2;
```

```
    end;
```

```
begin
```

```
    x := 3;
```

```
    P2;
```

```
end;
```

What is the output of P1 when static scoping is employed?

What is the output of P1 when dynamic scoping is employed?

15. When is a named constant bound to its value?

Chapter 6 Review Questions

1. What are the standard primitive data types? Are they the same for every programming language?
2. What is dynamic typing? Static typing?
3. Describe the internal representations commonly used to store unsigned integer values, signed integer values, real values, and character values.
4. What design issues should be considered when including character strings in a programming language?
5. Describe the commonly used strategies for storing character strings. How do C and C++ strings differ?
6. What is an enumeration type? What benefits are derived from using enumerated types?
7. What is a subrange type?
8. What design issues should be considered when including arrays in a programming language?
9. What are the 4 categories of subscript binding and storage allocation used for arrays?
10. Why might a language limit the number of subscripts for an array?
11. What array operations are implemented in C++? In Pascal?
12. What is an array slice? What is its purpose?
13. What is an array mapping function?
14. Given the following Pascal declaration for array A, if the base address of A is 1024, where is A[3][4] located? (Allow 2 bytes per integer value)
`var A : array [-10..10][0..5] of integer;`
15. Given the following FORTRAN 90 declaration for array A. if the base address of A is 1024, where is A(3,4) located? (Allow 2 bytes per integer value)
`INTEGER A (1..10, 1..5)`
16. Given the following C/C++ declaration for array A. if the base address of A is 1024, where is A[3][4] located? (Allow 2 bytes per integer value)
`int A[10][10];`
17. What advantage is gained by using 0 relative indexing for arrays?
18. What is an associative array?
19. What is a record type? What programming language first introduced the concept of records?
20. What types of operations can be performed on a Pascal record? On a C/C++ struct?
21. How are the fields of a record stored? How are they accessed?
22. What is a union type? What are the advantages and disadvantages of union types? How does Ada improve upon the Pascal union type?
23. What is a set type? What operations can be performed on sets?
24. What is a pointer type? What operations can be performed on pointers?
25. What problems can be caused by the improper uses of pointers?
26. How are pointers and arrays related in C/C++?
27. What is a reference type?
28. What are the 2 commonly implemented heap management techniques? What are the benefits and drawbacks of each?
29. Describe the differences between the reference counter and garbage collection approaches to heap management.

Chapter 7 Review Questions

1. How do the operator precedence rules used by C++ differ from those used by C?
2. What associativity rules are used by C/C++? How do the rules used by Ada differ from those used by FORTRAN?
3. What is operand evaluation order?
4. How do side effects affect the result of an arithmetic operation?
5. What is operator overloading? How is operator overloading employed in C and Pascal? In C++?
6. What is coercion?
7. How is the type of a mixed mode arithmetic expression determined by C/C++?
8. How is explicit type conversion implemented in C/C++?
9. What relational operators are used by C? Pascal? Smalltalk?
10. What is a Boolean expression?
What is the result of the C/C++ expression $(5 < 10) \ \&\& \ (7 > 1)$?
What is the result of the Pascal expression $(5 < 10) \ \text{OR} \ (7 > 1)$?
11. What is Short-Circuiting?
12. What is the result of the C/C++ statement $a = b = c * 2$?
13. What is the result of the C/C++ statement $a = b < c ? b : c$?
14. Illustrate how a queue and stack will be used to generate intermediate code to evaluate the C/C++ statement $a * b - c + d * 2 - e$.
15. Illustrate how queues and stacks will be used to generate intermediate code to evaluate the C/C++ statement $a * b - (c + d * 2) - e$.
16. Illustrate how a queue and stack will be used to generate intermediate code to evaluate the APL statement
 $A * B - C + D * 2 - E$.

Chapter 8 Review Questions

1. What is a control structure? What are the 3 standard categories of control structures?
2. What is a compound statement? What is its purpose? What language first introduced the concept of compound statements? How is a compound statement created in C/C++? In Pascal? In Smalltalk?
3. What is a selection statement? What types of selection statements are included in C/C++? In Pascal? In Smalltalk?
4. Differentiate between 2-way selection statements, 3-way selection statements, and multiple selection statements.
5. What benefits are gained by including an “else if” in a programming language?
6. What multiple selection statements are implemented in C/C++? Pascal? BASIC? FORTRAN I?
7. How are C/C++ switch and Pascal case statements similar? In what ways do they differ?
8. Draw flowcharts for each of the following Pascal code segments:
IF a < b THEN IF c < d THEN e := a + c ELSE e := b + d;
IF a < b THEN IF c < d THEN e := a + c; ELSE e := b + d;
9. What is an iterative statement? What types of iterative statements are included in C/C++? In Pascal? In Smalltalk?
10. What counting loop construct was provided by Algol 60 that is NOT seen in today’s programming languages?
11. How are the FORTRAN 77 and FORTRAN 90 DO loops different from those seen in previous versions of FORTRAN?
12. Describe the 3 segments of C/C++/Java for loop header. What does the following C/C++/Java for loop do? for (;;);
13. How does the C/C++/Java for loop structure differ from the for loop structure seen in Pascal, Ada, or Smalltalk? How do all of these for loop structures differ from the FORTRAN 77 DO loop model?
14. What is a user-controlled loop mechanism? What benefits are derived from user controlled loops?
15. What is an unconditional branch statement? Why is the use of unconditional branching discouraged in today’s software development environments?
16. Differentiate between C/C++ continue, break, return, and exit statements. How do C++ and Java break statements differ?
17. What is a guarded command?
18. How does the Bohm and Jacopini conjecture of 1966 affect today’s programming languages and styles?

Chapter 9 Review Questions

1. What are the general characteristics of a subprogram?
2. Differentiate between a subprogram's definition, header, and call. Give an example of each.
3. Differentiate between formal and actual subprogram parameters. Give an example of each.
4. Differentiate between positional and keyword parameters. Give an example of each.
5. Differentiate between functions and procedures. Give an example of each.
6. What issues must be considered regarding subprograms when designing a new programming language?
7. What is a local referencing environment?
8. Differentiate between the 5 different parameter passing models? What parameter passing models are used in FORTRAN? C? C++? Pascal? Smalltalk? Scheme? Prolog?
9. What is a thunk? What is its purpose?
10. How is the run-time stack related to subprogram execution?
11. How are arrays passed in C/C++? FORTRAN? Pascal? What are the benefits and drawbacks of each of these array passing techniques?
12. What issues must be considered when selecting the array passing models to be implemented in a programming language?
13. How are subprogram names passed as parameters in C/C++? What are the benefits and drawbacks of passing subprogram names? Create a C/C++ code segment that illustrates passing a function with no parameters. Create a C/C++ code segment that illustrates passing a function with one integer parameter.
14. What is subprogram overloading? Give an example of subprogram overloading in C++. What issues must be considered when designing overloaded subprograms? How do overloaded subprograms differ from overloaded operators?
15. What is a generic? Give an example of a generic C++ subprogram. What issues must be considered when designing generic subprograms?
16. What is separate compilation? What are the benefits of separate compilation?
17. What is independent compilation? What are the benefits of independent compilation?
18. Why can't templates be separately compiled?
19. What is an external definition? How are references to external subprogram definitions handled in C and C++?
20. What is a subprogram side effect?
21. What return types are permitted by C? By C++? By Pascal?
22. What non-local values can be accessed by a C/C++ subprogram? By FORTRAN subprograms? What is dynamic scoping?
23. What is a co-routine?

Chapter 10 Review Questions

1. What steps must be followed when calling a subprogram?
2. What steps must be followed when returning from a subprogram?
3. What is an activation record? What is its purpose? What does it contain?
4. How are activation records related to recursion?
5. Given the following recursive Prolog rule, trace the activation records that are created for the query: `concat(X,Y,[1,2,3,4,5])`.
`concat ([],List,List).`
`concat ([Head|Tail1],List,[Head|Tail2]) :- concat(Tail1,List,Tail2).`
6. Describe how static chains or displays are used to implement dynamic scoping.
7. Differentiate between shallow and deep access when accessing non-local variables.

Chapter 1 Review Questions

1. What is abstraction? What is encapsulation? In the context of Object Oriented Programming and Design, how are abstraction and encapsulation related?
2. What is an Abstract Data Type? What support for abstract data types is provided by C++? By Pascal? By Smalltalk? By Ada? By Modula-2?
3. What language first introduced the concept of Object Oriented Programming?
4. Differentiate between C++ `public`, `private`, and `protected` entities.
5. What is a constructor? A destructor? When are constructors called? When are destructors called? Why does Java not have destructors?
6. What is a Parameterized Abstract Data Type? What support for parameterized abstract data types is provided by Ada? By C++?
7. Define a C++ Template class called `CoordPair` that can hold a pair of values of the same type. Create a constructor, a destructor and an output operator (`<<`) for this class.

Chapter 2 Review Questions

1. What does the term inheritance mean in the context of Object Oriented Design?
2. What is Polymorphism? Dynamic Binding? How are they related?
3. How do Java's and C++'s implementations of Object Oriented Programming differ?

Chapter 3 Review Questions

1. What is the goal of concurrent processing?
2. What is a multi-processor architecture? What is SIMD? MIMD?
3. What is a semaphore? How are semaphores employed by concurrent processes?
4. What is threading?

Chapter 14 Review Questions

1. What is an exception?
2. Describe 3 common exceptions.
3. What is an exception handler?
4. Differentiate between termination and continuation with respect to exceptions.
5. What language pioneered the implementation of user defined exception handlers?

Chapter 15 Review Questions

1. Describe the functional programming paradigm.
2. How do functional and imperative programming languages differ?
3. How do functional languages relate to mathematical functions?
4. What is referential transparency?
5. What was the first functional language?
6. What is a lambda function? How are lambda functions used?
7. What is a functional form?
8. What is function composition? Construction? Apply-to-all? How are each of these implemented in Scheme?
9. What is a predicate function?
10. List 4 functional languages.

Chapter 16 Review Questions

1. What is a declarative language?
2. What is the Predicate Calculus? What is a Proposition? A Clausal Form?
3. Describe the relationship between predicate calculus and logic programming languages.
4. What is meant by the term “declarative semantics”?

Scheme Review Questions

1. Define each of the following with respect to Scheme:
 - a) atom
 - b) primitive
 - c) lambda expression
 - d) free variable
 - e) top level definition
 - f) predicate
 - g) type predicate
 - h) lexical scoping
 - i) thunk

2. Describe the function and use of each of the following Scheme primitive functions:
 - a. list?
 - b. number?
 - c. null?
 - d. cddddr
 - e. cadr
 - f. if
 - g. let

3. You should be able to read and interpret a segment of Scheme code.

4. You should be able to write a segment of Scheme code.

Prolog Review Questions

- 1) Given the following recursive Prolog rule, trace the activation records that are created for the query: `concat(X,Y,[1,2,3,4,5])`.
 - i) `concat([],List,List)`.
 - ii) `concat([Head|Tail1],List,[Head|Tail2]) :- concat(Tail1,List,Tail2)`.
- 2) What is a declarative language?
- 3) What is the Predicate Calculus? What is a Proposition? A Clausal Form?
- 4) How does a Prolog fact differ from a Prolog query?
- 5) What is a Prolog rule statement?
- 6) What is a Prolog goal statement?
- 7) Describe the Prolog inferencing process.
- 8) Under what conditions can the Prolog verb `is` be used for assignment?
- 9) Describe the Prolog list structure.
- 10) Describe how are the following symbols are used in Prolog: `.`, `;`, `=`
- 11) What is a Prolog world?
- 12) What is a finite domain solver?
- 13) What applications of Prolog are described by the author of your book.
- 14) What limitations of Prolog are described by the author of your book?
- 15) You should be able to write a segment of Prolog code. For example:

The Prolog facts listed here partially describe the small family tree illustrated below.

- ```

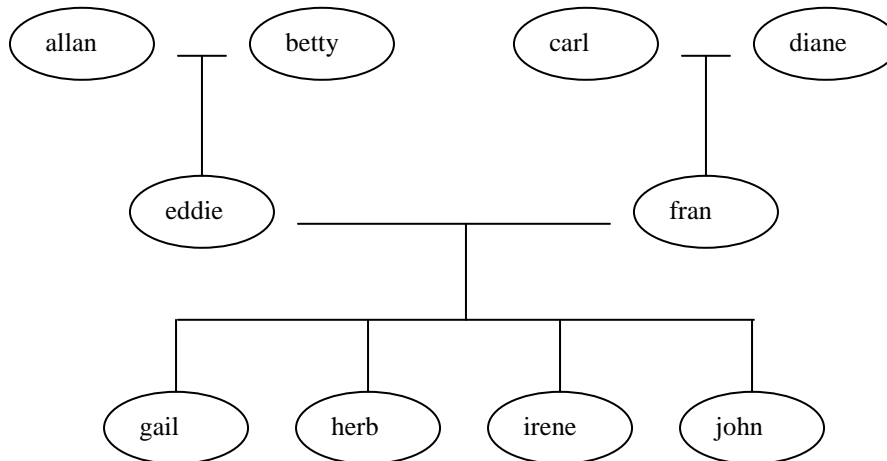
father(allan,eddie).
mother(diane,fran).
father(eddie,gail).
mother(fran,john).

```
- a. Complete the definition of this family tree.
  - b. Write a rule that indicates that X is a parent of Y if X is either the mother or father of Y.
  - c. Write a rule that indicates that X is a grandparent of Y if X is a parent of Z and Z is a parent of Y.
  - d. What is the complete output of each of the following queries:
 

```

[? grandparent(carl,herb).
[? grandparent(GP,irene).
[? grandparent(betty,GC).
[? grandparent(GP,GC).

```



- 16) To what programming paradigm family does the language Prolog belong?
- 17) For what types of applications is Prolog an appropriate programming language?
- 18) Describe each of the following terms with respect to the Prolog programming language:
  - a) world
  - b) fact
  - c) rule
  - d) base
  - e) predicates
  - f) arguments
  - g) arity
  - h) list
- 19) Describe the Prolog syntax rules regarding each of the following:
  - a) selecting variable names
  - b) defining constant (literal) values
  - c) use of periods (.)
  - d) use of semicolons (;)
  - e) use of commas (,)
  - f) formulation of facts
  - g) formulation of queries
  - h) formulation of rules
  - i) assignment statements
  - j) conditional expressions
- 20) How do you create a Prolog source file? How do you document a Prolog source file? How do you load the file when using the Prolog interpreter?
- 21) What is a Prolog “list”?
- 22) What is the syntax used to create a Prolog list?
- 23) How are the terms “head” and “tail” defined with respect to a Prolog list?
- 24) How is membership in a list determined?
- 25) How is an empty list represented?

# Java Review Questions

1. Java is an Object Oriented Language. However, it is not completely object oriented. Why?
2. How does the inheritance model employed by Java differ from that used in C++? How does the Java compiler determine that a class cannot have derived classes?
3. An integer value can be declared in Java as an “int” or as an “Integer”. Describe the differences between these two methods. When would you use each?
4. What is Swing? What resources are provided by Swing for creating windows? What resources are provided for using mouse input in an application program.
5. What is the function of the PaintComponent method? When is the PaintComponent method called?
6. What is dynamic binding?
7. While the “new” operator is frequently used in Java, the core language does not support a “delete” operator. Why? What techniques are used to avoid memory overflow?
8. Write a segment of Java code to declare and initialize an array of N integer values.

## Thoughts on Computer Programming Languages

The use of COBOL cripples the mind; its teaching should, therefore, be regarded as a criminal offence. (Edsger Dijkstra)

Consistently separating words by spaces became a general custom about the tenth century A.D., and lasted until about 1957, when FORTRAN abandoned the practice. (Sun FORTRAN Reference Manual)

Cobol has almost no fervent enthusiasts. As a programming tool, it has roughly the sex appeal of a wrench. (Charles Petzold)

C++ is the only current language making COBOL look good. (Bertrand Meyer)

C++ has its place in the history of programming languages. Just as Caligula has his place in the history of the Roman Empire. (Robert Firth)

Arguing that Java is better than C++ is like arguing that grasshoppers taste better than tree bark. (Thant Tessman)

Java is, in many ways, C+++-. (Michael Feldman)

If Java had true garbage collection, most programs would delete themselves upon execution. (Robert Sewell)

It is practically impossible to teach good programming style to students that have had prior exposure to BASIC; as potential programmers they are mentally mutilated beyond hope of regeneration. (Edsger Dijkstra)

In my egotistical opinion, most people's C programs should be indented six feet downward and covered with dirt. (Blair P. Houghton)

C++ is history repeated as tragedy. Java is history repeated as farce. (Scott McKay)

Unix and C are the ultimate computer viruses. (Richard P Gabriel)