CSCI E-95, Fall 2023: Compiler Design and Implementation

Prof. James L. Frankel
Harvard University

Version of 6:29 PM 12-Dec-2023

First Class Meeting on 9/5/2023

First Class Meeting Agenda

- Class website & Canvas, Zoom links
- Staff introductions
- Polls
- Student introductions
- Class website & class information
- Student actions: Order books, Say Hello!, Student Locations, HarvardKey, Cisco AnyConnect VPN, cscie95.dce.harvard.edu instance
- Course problem set overview
- Problem Sets 0 & 1
- Overview of Compiler & Review of the C Programming Language

Class Website, Canvas, Zoom Links

- Our **class website** is located at URL: **https://cscie95.dce.harvard.edu/fall2023/**

- Please participate in the live stream and ask questions verbally using **Zoom** available in **Canvas** (**https://canvas.harvard.edu/courses/131417**) under the **Zoom** menu
- In addition, questions may be asked textually using **Zoom's Chat facility**

Zoom

- You are encouraged to turn on your video feed
  - This allows the course staff to better determine if students seem puzzled and/or if they have questions

- Class and section meetings are recorded
  - Students who are unable to attend a meeting for any reason are able to view recordings later
  - It's still better to participate in the live session so that questions can be asked and answered
  - Many students find that reviewing material later to fully appreciate the details presented during class – even if they participated in the live class – is very helpful

Staff Introductions

- Professor
  - James “Jamie” Frankel
- Teaching Assistants
  - Daniel Willenson
  - Mark Ford

Quick Polls

- Goals and Interest for Class Enrollment (Multiple Choice)
- Class Participation
- Section Participation
- Class Expectations (Multiple Choice)

- You can choose to answer the polls anonymously

Student Introductions

- Please tell us a little about yourself
  - Where you're located
  - What you do when you're not at Harvard
  - Your technical background
  - Your out-of-work/school hobbies

Tour of Class Website

- At the top there are alerts in red
- Quick Links
- Links for streaming and videos
- Info about midterm exam, prerequisites, learning objectives, overview, bibliography, instructors and section, Ed Discussion wiki/forum, Say Hello!, your location, git & GitHub, grading, accessibility, plagiarizing, use of generative AI, publishing/distributing course materials, MIPS & SPIM, outline/approximate schedule, agenda for the upcoming class, slides used in class, questionnaire & problem sets, assorted links, example programs used in class, grammar for our C Programming Language, link to the section home page

Meeting Times

- Section meets on Tuesdays from 6:45 PM to 7:45 PM Eastern Time (ET) and in Zoom using the **Section: Helix Classroom** room
  - This is immediately before class meets

- Class meets on Tuesdays in Room L01, 53 Church Street, Harvard Square, Cambridge, Massachusetts from 8:00 PM to 10:15 PM Eastern Time (ET) and in Zoom using the **Class: Helix Classroom** room
  - Elongated class meeting time

- I will attempt to include a break during the class meetings *(but no guarantee because of scope of material to be presented)*

Section

- Required part of class

- Very important
  - Discusses concepts & issues that are not covered in class
  - Often gives a sketch of algorithms and approaches to be used in solving the problem sets
  - Adds enrichment on topics discussed in class/lecture

  - Great forum for a more interactive dialog

  - Is live streamed and also recorded

Class Website Review

- Questions?
  - Questions are always welcomed
    - Any questions now?
  - If there is limited time to answer a questions, I'll let you know

- Review of Website
  - Midterm exam
  - Prerequisites
  - Overview
  - Required and optional books
  - The daily agenda (these slides) and all slides used in class
  - …

- Order books, if you have not already done so
  - **Compilers: Principles, Techniques, and Tools**, **2nd Ed.** by Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman
  - **C: A Reference Manual**, **5th Ed.** by Samuel P. Harbison and Guy L. Steele, Jr.

- Somewhat limited online access is available to all of our books through our Library Reserves link in Canvas

Required Readings

- Refer to the Approximate Schedule section of the course website for required readings to be completed before each class meeting

Say Hello!, Student Locations, Harvard Key, Using cscie95.dce.harvard.edu

- Submit a video in Canvas under **Discussions** as a reply to my “Say Hello!” topic
- Please post your primary location using **Student Locations** facility in Canvas
- Ensure that your **Harvard Key** is established
- Ensure that you are able to VPN into Harvard using vpn.harvard.edu and Cisco AnyConnect
- Ensure that you have an account on our cscie95.dce.harvard.edu AWS instance
  - **Once your VPN connection is established**, login to cscie95.dce.harvard.edu using SSH/SFTP (SecureCRT & SecureFX) with your HarvardKey NetID as your login name and your HarvardKey password as your password
  - If you are unable to login to cscie95.dce.harvard.edu, you may need to synchronize your HarvardKey password by using a browser to visit https://key.harvard.edu/manage-account and then clicking on “Synchronize Password >” and following the instructions on the next screen

g.harvard.edu e-mail Address

- If you're interested, you can get a g.harvard.edu account that will give you a *logname*@g.harvard.edu e-mail address and access to Google Apps for Harvard
  - Get started at http://g.harvard.edu/
  - Note: Please be aware that when you claim your g.harvard account that g.harvard will become your primary Harvard e-mail address. All official communication from Harvard will be sent to your new g.harvard address and your g.harvard account will become your HarvardKey login name.

Class Discussion Group: Ed Discussion

- Ask all non-personal questions in Ed so the whole class can benefit from the answers
  - Ed can be found in Canvas by following the Ed Discussion link (https://canvas.harvard.edu/courses/131417/external_tools/98387?display=borderless)
  - Students are welcome to answer questions there, too
  - Personal questions should be sent to the course staff via e-mail
    - If appropriate, include all three course staff members in e-mails to allow the fastest reply
  - All registered students should already be in our Ed group

- To **receive immediate notifications about new threads:** in our Ed group, in the upper right, click on the **Account** icon, then…
  - Select **Settings**
  - Click on **Notifications**
  - Change the frequency to receive e-mails about new threads to **Instant**
  - Also, ensure that all other **Notification Emails** are active

Midterm Exam

- Our midterm exam will be available starting at 8:00 PM ET on October 24, 2023
  - The exam must be started within 24 hours of the date & time above
  - The exam is three hours in length
  - The exam will be administered under Proctorio
- There will be **no class meeting on October 24th**, but **section will still be held on October 24th**
  - No topics relevant to the midterm exam will be discussed in that section meeting

Problem Set Overview

- Problem Set 0: the course questionnaire, fix-this-program & word-count
- Problem Set 1: lexer for our C Programming Language subset
- Problem Set 2: parser for our compiler
- Problem Set 3: symbol table management system
- Problem Set 4: semantic analysis
- Problem Set 5: IR generation
- Problem Set 6: MIPS assembly language code generation
- Final Project: optimizations

Five Free Late Days

- Please don’t use any of your five free late days early in the class
- Because the later problem sets are built upon earlier problem sets, the free late days are more valuable later in the semester
- Also, the larger problem sets are worth more points and take much more time to complete

Problem Set 0

- Complete Problem Set 0
  - Establish a GitHub account
  - Install git as described on the section website (https://cscie95.dce.harvard.edu/fall2023/section/index.html)
  - Modify the course questionnaire with your personal answers
  - Fix warnings and errors in fix-this-program on the cscie95 instance
  - Write the word count program
  - Create a branch named "problem-set-0", create a merge request, add the appropriate comment
- Due this coming Sunday night, September 10, 2023 at midnight ET

Problem Set 1

- Present **Problem Set 1**
  - Due at midnight ET on Sunday night, September 17th, 2023

- The code of your compiler can take advantage of all of the syntax and features of the latest ISO/ANSI C Programming Language

Lying to Students

- I will lie to you this semester

Lying to Students

- I will lie to you this semester
  - There are too many details to give the whole truth
  - That is the only way we can make reasonable progress through the material

- By the end of the semester, all lies will be fully corrected

Non-academic Class Activities

- Encourage a student community

- If interested, students are welcome to gather with us after each class for dinner
  - Opportunity for students to socialize in an informal setting outside of class
  - Discussion/conversation/sharing after class
  - Not relevant to class

- Other non-class activities
  - Class ski trip in January
  - Sailing trip during the summer

Class Break

- Let’s take a 10 minute break
- You’re welcome informally interact during the break

Today's New Material

- Cover **Overview of Compiler** slides
- Cover **Review of the C Programming Language** slides through the **Language: Operators** slide except for the **Observation**

Second Class Meeting on 9/12/2023

- Questions?
  - Problem Set 0
  - Problem Set 1
  - Section or lecture
  - Anything else

Problem Set 1

- Keep code in src/compiler directory
- Accept the same command line interface as does the skeleton code in the class repository
- You are not required to keep the same structure that is used by the skeleton compiler, but it is probably the correct approach unless there is a good reason to deviate from its structure
- The link to Manual for Lexical Analysis with Flex, Flex 2.6.2 on the class website under **Open software and documentation** has a great deal of useful information

- Reminder that **Problem Set 1 is due at midnight ET on Sunday night, September 17, 2023**

Problem Set 2

- Present **Problem Set 2**
  - Due at midnight ET on Sunday night, October 1st, 2023
- Show the grammar for our subset of the C Programming Language on the class web site

- **Problem Set 2 is perhaps the most time-consuming problem set in the course**
  - Start early
  - Ask questions

- We'll present information on yacc/bison next week

New Material for this Week (1 of 3)

- Cover new slides
  - **Lexical Analysis**
- Example
  - Construct an NFA from a regular expression using the McNaughton-Yamada-Thompson algorithm ***exactly***
  - Construct a DFA from an NFA using the Subset Construction
  - Cover new slides
    - **Example of Regex, NFA, DFA**

New Material for this Week (2 of 3)

- Cover new slides
  - **Using Lex**
- Example
  - Show **lexer-standalone.lex** & **lexer.c**
  - Run **make standalone**
  - Demonstrate **./lexer**

New Material for this Week (3 of 3)

- Cover more new slides
  - **Review of the C Programming Language**
    - Continue with the **Language: Operators** slide with the **Observation** through the **end of that Language: Operators** slide

Third Class Meeting on 9/19/2023

Third Class Meeting Agenda

- Questions and Comments
- Administrivia
  - Start Work on Problem Sets Early
  - Section
  - Lectures & Section Meetings
  - Overall Compiler Directions
- Aho, Lam, Sethi, Ullman Compilers Textbook
- Review of Current Problem Set Status
- Syntax Analysis
  - Show **recursiveDescentParser.c**
- Using YACC/Bison
  - Show **lexer.lex** & **parser.y**
- Parse Tree, AST, and Type Tree
- Review of the C Programming Language (continued)

Questions and Comments

- Section
- Last week's class or any previous classes
- Problem Sets 0 through 2
- Using the class cscie95.dce.harvard.edu instance
- Ed Discussion threads
- Readings
- Anything else

Start Work on Problem Sets Early

- In order to ensure that your questions are answered in a timely manner, start work on problem sets early
- The course staff are often quite prompt in answering Ed questions, but there is no guarantee of immediate responses

Section

- Required part of class
- Section meets immediately before class on Tuesdays from 6:45 PM to 7:45 PM Eastern Time (ET) and in Zoom using the **Section: HELIX Classroom** room

- Very important
  - Discusses concepts & issues that are not covered in class
  - Often gives a sketch of algorithms and approaches to be used in solving the problem sets
  - Adds enrichment on topics discussed in class/lecture

  - Great forum for a more interactive dialog

  - Is live streamed and also recorded

Lectures & Section Meetings

- Watch both the lecture and section videos
- All lecture and section videos are recorded
- Many questions are answered in lecture and section

Overall Compiler Directions

- Base your compiler on the skeleton compiler in the class repository
- Be sure to support the required command line interface
- The Problem Set descriptions are very detailed
  - They try to answer all questions and fully-specify the assignment
- Follow all posts and replies on Ed
  - They will answer questions about class and problem sets

Aho, Lam, Sethi, Ullman Compilers Textbook

- Read the assigned textbook readings
- Do the practice problems in the textbook
- Even though we are not assigning textbook problem sets, you are responsible for the readings
  - Useful for the programming problem sets
  - Necessary for the midterm exam

Problem Set 1

- **Problem Set 1**
  - Was due at midnight Eastern Time on Sunday, September 17th, 2023

- Everyone should have already completed PS1

Problem Set 2 (1 of 2)

- **Problem Set 2**
  - Due at midnight Eastern Time on Sunday, October 1st, 2023

- Problem Set 2 is the most time-consuming problem set in the course
  - Start early
  - Ask questions

Problem Set 2 (2 of 2)

- Every node in the AST should have a node type
  - Which operator/statement?
  - Which/how many children?
  - etc.
- Don't create unnecessary nodes
  - No node for parenthesized expression
  - No node for a choice of non-terminals (such as, a or b or c)
  - No node for terminals (in most cases)
  - etc.
- Create nodes to store identifiers as strings in the AST
- Create nodes to store literals as values in the AST
  - Integer literals as binary integral values with type
  - String literals as strings

New Material for this Week (1 of 4)

- Cover new slides
  - **Syntax Analysis**
- Example
  - Show **recursiveDescentParser.c**
  - Run **make recursive**
  - Demonstrate **./recursiveDescentParser**

New Material for this Week (2 of 4)

- Cover new slides
  - **Using YACC or Bison**
- Example
  - Show **lexer.lex** & **parser.y**
  - Run **make parser**
  - Demonstrate **./parser**

New Material for this Week (3 of 4)

- Cover new slides
  - **Parse Tree, AST, and Type Tree**
  - Just look at the Parse Tree and AST portions of these slides (*i.e.,* ignore the Type Tree slide for now)

New Material for this Week (4 of 4)

- Cover more new slides
  - **Review of the C Programming Language**
    - Continue from the **Unary, Binary, and Ternary Operators** slide #6 through the **Details of the Binary Infix Logical Operators** slide #15

Fourth Class Meeting on 9/26/2023

Fourth Class Meeting Agenda

- Questions and Comments
- Correction from Last Week's Class
- Administrivia
  - Due Date for PS2
- Current Problem Set Status
  - PS2
  - Present PS3
- Symbol Table Management
- Parse Tree, AST, and Type Tree
  - Present the Type Tree
- Type Checking
- Review of the C Programming Language (continued)

Questions and Comments

- Section
- Last week's class or any previous classes
- Problem Sets 0 through 2
- Lexer
- Parser & AST construction
- Our grammar
- Review of the C Programming Language
- Ed Discussion threads
- Readings
- Anything else

Correction from Last Week's Class

- Two examples used in class last week
  - k = i++, j++;
  - k = (l = i++), (i + j++);

- Part of our discussion was incorrect because the precedence of the comma (sequential evaluation) operator is lower than the precedence of the assignment operators – in fact, the comma operator has the lowest precedence of all C operators (see H&S Page 205, Table 7-3)
  - Therefore, the value assigned to k is the result of evaluating the i++ expression and is not affected by the right operand of the comma operator
  - The value of the entire comma expression is the result of evaluating the right operand of the comma expression, but that result is not used; therefore, the second example is parenthesized as:
    - (k = (l = (i++))), (i + (j++));

- This expression assigns to m the result of evaluating the right operand of the comma operator:
  m = (k = (l = i++), (i + j++));

Due Date for PS2

- The due date for PS2 has been extended one week to now be due at midnight ET on Sunday, October 8th, 2023

- Unfortunately, however, this does not affect the due date for PS3 which is one week later
  - So, still complete PS2 as soon as possible

Problem Set 2

- **Problem Set 2**
  - Now due on Sunday, October 8th, 2023 at midnight Eastern Time
  - You should all have already made significant progress working on PS2

- Problem Set 3 is based on the AST from PS2

Problem Set 3

- Present **Problem Set 3**
  - Due Sunday, October 15th, 2023 at midnight Eastern Time

- Problem Set 3 involves building symbol tables & type data structures

New Material for this Week (1 of 4)

- Cover new slides
  - **Symbol Table Management**
  - Go over symbolTables.c

New Material for this Week (2 of 4)

- Cover new slides
  - **Parse Tree, AST, and Type Tree**
  - Let’s examine the Type Tree slide
  - See how the Type Tree is derived from the AST for a decl

New Material for this Week (3 of 4)

- Cover new slides
  - **Type Checking**
  - Cover through the **Minimum Integer Precision and Range (§5.1.1, p. 125 & Table 5-2, p. 127)** slide

New Material for this Week (4 of 4)

- Continue with the C Programming Language slides
  - **Review of the C Programming Language**
    - Continue from the **Ternary Infix Operator** slide #16 through the **Lvalues vs. Rvalues (part 1 of 2)** slide

Fifth Class Meeting on 10/3/2023

Fifth Class Meeting Agenda

- Questions and Comments
- Current Problem Set Status
  - PS2
  - PS3
- Upcoming Midterm Exam
- Symbol Table Management
- Table-Driven Top-Down Parsing
- Type Checking
- Review of the C Programming Language (continued)

Questions and Comments

- Section
- Last week's class or any previous classes
- Problem Sets 0 through 3
- Lexer
- Parser & AST construction
- AST, Type Tree, Symbol Table
- Our grammar
- Review of the C Programming Language
- Ed Discussion threads
- Readings
- Anything else

Problem Set 2

- **Problem Set 2**
  - Now due on Sunday, October 8th, 2023 at midnight Eastern Time

- How is everyone doing with PS2?
- Can you accept a small snippet of our C code in your compiler and emit the resulting AST using your pretty-printer that produces the same code when passed through your compiler once again?

Problem Set 3

- **Problem Set 3**
  - Due Sunday, October 15th, 2023 at midnight Eastern Time

- Problem Set 3 involves building symbol tables & type data structures
  - Symbol tables
    - Other names
      - Identifier resolution to entry in correct symbol table
      - Replace identifier in AST with pointer to entry in correct symbol table
    - Statement labels
    - String table
  - Type tree
    - Each other names symbol table entry must have an appropriate type tree attached to it
  - Create all needed pointers to allow traversal both inward and outward among symbol tables

Upcoming Midterm Exam

- Midterm Exam
  - Three weeks from today

  - Earliest start time is 8:00 PM ET on Tuesday, October 24th, 2023
  - Must be started within 24 hours of 8:00 PM ET on Tuesday, October 24th, 2023
  - Must be completed within 3 hours of the time the student starts the exam

  - Administered by Proctorio/Canvas
    - No books, notes, computers, etc.
    - Canvas needed for access to slides

  - Unfortunately, may ***not*** be taken in class
    - Consistency for all students
    - Controlled access to slides
    - Building closes at 11 PM

Inward/Downward Edges among **Symbol Tables**

- Inward/Downward edges are useful for printing all symbol tables and for performing other operations
  - For example, we will need to traverse all symbol tables within each procedure/function to:
    - Determine the size of the stack frame required
    - Determine the stack offset for each automatic (stack-based) variable

Outward/Upward Edges among **Symbol Tables**

- Outward/Upward edges are useful for searching through symbol tables for the appropriate scope in which an identifier is declared
  - For example, when encountering a reference to an identifier, a search must be conducted starting with the current block to outermore blocks, to the procedure scope, and finally to the file scope to find the declaration for the identifier

Outward/Upward Edges in **AST**s

- Upward/Outward edges are useful in the AST when performing some operations in your compiler
  - For example, searching for the innermost matching language construct for **break** and **continue** statements

New Material for this Week (1 of 3)

- **Table-Driven Top-Down Parsing**

New Material for this Week (2 of 3)

- Type checking slides
  - **Type Checking**
  - Continue after the **Minimum Integer Precision and Range (§5.1.1, p. 125 & Table 5-2, p. 127)** slide

  - Skip over the **Numeric Encodings** slides
  - Skip over the **Character Encodings** slides

New Material for this Week (3 of 3)

- Continue with the C Programming Language slides
  - **Review of the C Programming Language**
    - Continue from the **Lvalues vs. Rvalues (part 1 of 2)** slide #26 through the **Visibility (§4.2.2)** slide #34

Sixth Class Meeting on 10/10/2023

Sixth Class Meeting Agenda

- Questions and Comments
- Harbison & Steele Readings
- Current Problem Set Status
  - PS2
  - PS3
  - PS4
- Upcoming Midterm Exam including Proctorio
- Upcoming Dates
- Type Checking
  - Signed and unsigned types in a comparison operator
- Complete the **Review of the C Programming Language** slides (continued)

Questions and Comments

- Section
- Last week's class or any previous classes
- Problem Sets 0 through 4
- Lexer
- Parser & AST construction
- AST, Type Tree, Symbol Table
- Our grammar
- Review of the C Programming Language
- Ed Discussion threads
- Readings
- Anything else

Harbison & Steele (1 of 2)

- Even though no specific readings from **Harbison & Steele** are assigned in the syllabus, everyone should be reading and referring to H&S on a regular basis

- Familiarity with Chapters 1, 2, and 4 through 9 is required and assumed

Harbison & Steele (2 of 2)

- The grammar is not sufficient to determine if a program is valid
- There are many examples where the grammar accepts invalid programs
- Each section of H&S contains constraints that must be checked to ensure that a valid program exists

Problem Set 2

- **Problem Set 2**
  - Was due this past Sunday, October 8th, 2023 at midnight Eastern Time

- Problem Set 2 involves building the AST

Problem Set 3

- **Problem Set 3**
  - Will be due this coming Sunday, October 15th, 2023 at midnight Eastern Time

- As you know, Problem Set 3 involves building symbol tables & type data structures

Problem Set 4

- **Problem Set 4**
  - Will be due Sunday, October 29th, 2023 at midnight Eastern Time
- Problem Set 4 involves semantic analysis/type checking
- Present **Problem Set 4**

Midterm Exam

- Our **Midterm Exam begins in two weeks** on October 24th, 2023 at 8:00 PM ET
  - There will be **no class meeting on October 24th**, but **section will still be held on October 24th**
    - No topics relevant to the midterm exam will be discussed in that section meeting

- Ask any questions relevant to the midterm exam in section, in class, in office hours, or on Ed before noon ET on the day of the exam

Midterm Exam

- Exam is **three hours in duration**
  - The exam is designed to take two hours
  - But, the exam is time-consuming – don't worry if you take all three hours
  - The problems are scored in points where a point is weighted to be approximately one minute of exam answer duration
- Earliest starting time is 8:00 PM ET on Tuesday, October 24th, 2023 – **two weeks from today**
- Latest starting time is 7:59 PM ET on Wednesday, October 25th, 2023
- The exam will be available through **Canvas** under the **Quizzes** tab
  - The exam will be administered under **Proctorio**
  - **Either Google Chrome, Microsoft Edge, Brave, or Opera must be used as the browser** for the exam – only these browsers are acceptable to Proctorio
  - Proctorio requires that a **Proctorio Extension** be installed into your browser
- No books may be used during the exam, but all slides from class will be available through the **Files** menu item in Canvas (a link to the **Files** page of Canvas is in the exam Instructions)
  - Nothing else can be used during the exam: no notes, no electronics other than the computer being used for the exam, no cell phones, no communication
  - But, nothing else is needed!

The Proctorio Setup Quiz

- *As soon as possible,* take the **Proctorio Setup Quiz** that is available in **Canvas** under **Quizzes**
- Really **carefully read all of the instructions**
  - **The instructions are the same as for the real Midterm Exam**
  - **We are not able to deal with issues caused by not following the instructions!**
- Contact the course staff *as soon as possible* if you have any problems taking the **Proctorio Setup Quiz**

Midterm Exam Format

- Different question formats
  - Essay
    - Text box in which answer can be typed
  - File Upload
    - Create your answer using: (1) a drawing application on your computer, (2) a drawing app from a web site, or (3) a piece of paper whose scan or photo is submitted.
    - In any of these cases, you will need to ***upload your answer file***. You must ensure that you have uploaded any answer files before the termination of the exam time. **No files can be uploaded after time has expired.**
  - Multiple Choice
    - Choose the correct answer
  - Multiple Answers
    - Select all correct answers

Material on Midterm Exam

- Book readings
  - Aho, Lam, Sethi & Ullman, Chapters 1-6
  - Harbison & Steele
    - Familiarity with Chapters 1, 2, and 4 through 9 is required and assumed
- Material covered in class through – and including – next week's class meeting except for Intermediate Representation (if we get that far)
- Material covered in Problem Sets 1-3
- Discussions on Ed
- Material covered in section is beneficial, and is required

Topics for Midterm Exam (1 of 3)

- C Programming Language
- Flex/Lex
- Regex's
- Context-Free Grammars
- Ambiguity
- Left Recursion
- Left Factoring
- Bison/YACC
- The Grammar for Our Subset-C Language
- NFA, DFA
  - Creating an NFA from a Regex using the McNaughton-Yamada-Thompson Algorithm
  - Conversion of NFA to DFA using Subset Construction

Topics for Midterm Exam (2 of 3)

- Top-Down Parsing
- Recursive Descent Parsing
- Symbol Table Management
- Types in the C Programming Language
- Representing Types in Our Compiler
- Parse Tree, AST
- Name Spaces
- FIRST, FOLLOW, LL(1) Grammars, Predictive Parsing Table M, Table-Driven Predictive Parsing
- Number Representations

Topics for Midterm Exam (3 of 3)

- Type Checking
- C Standard Conversions
- Bottom-Up Parsing
- Shift-Reduce Parsing

Midterm Directions

- Read each question very carefully
  - The question should include all necessary information
- The point values in the exam are equal to the number of minutes that a student who would score highly might take to complete the question
- The exam in lengthy – don’t be concerned if you don’t complete everything
- Initially, spend approximately the number of minutes on each question that the question is worth, then return to the question later to complete them as available time permits

Reviewing Upcoming Dates

- PS3 Due: Sunday, October 15th, 2023 at midnight ET

- We have just one more class meeting (on Tuesday, October 17th, 2023) before the Midterm Exam
- And, we have just one more non-exam class meeting before PS4 is due

- Midterm Exam
  - Earliest start time is 8:00 PM ET on Tuesday, October 24th, 2023
  - Must be started within 24 hours of 8:00 PM ET on Tuesday, October 24th, 2023
  - Must be completed within 3 hours of the time the student starts the exam

- PS4 Due: Sunday, October 29th, 2023 at midnight ET

New Material for this Week (1 of 2)

- Type checking
  - Discuss what happens with:

```
int i = -5;
unsigned int ui = 6;

if(i < ui)
  printf("i < ui\n");
else
 printf("i >= ui\n");
```

New Material for this Week (2 of 2)

- Continue with the C Programming Language slides
  - **Review of the C Programming Language**
    - Continue from the **Extent (or Lifetime or Storage Duration) (§4.2.7)** slide #35 through the last slide

Seventh Class Meeting on 10/17/2023

Seventh Class Meeting Agenda

- Questions and Comments
- Administrivia
- Problem Set 3 Modification
- Harbison & Steele Readings
- Programs Accepted by our Grammar
- Current Problem Set Status
  - PS2
  - PS3
  - PS4
- **Proctorio**
- Next Week's **Midterm Exam**
- Upcoming Dates
- Issues with Cross-Compiling
- Cover the **Shift-Reduce Parsing** slides
- Cover the **Numeric Encodings** slides
- Cover the **Character Encodings** slides
- [Symbol tables for symbolTables.c]
- [System header files]
- Cover the **Intermediate Representation** slides

Questions and Comments

- Section
- Last week's class or any previous classes
- Problem Sets 0 through 4
- Midterm Exam
- Lexer
- Parser & AST construction
- AST, Type Tree, Symbol Table
- Our grammar
- Review of the C Programming Language
- Ed Discussion threads
- Readings
- Anything else

Administrivia

- Moved Wednesdays office hours to Thursday
- Our office hours that were scheduled for Wednesdays from 7 to 8 PM ET have been moved to Thursdays from 7 to 8 PM ET for the rest of this semester
- Mondays office hours are still from 8:30 to 9:30 PM ET

Problem Set 3 Modification

- The required command line switch to output annotated comments for each reference to an identifier has been modified
  - It was -annotate
  - It is now -a

- This allows getopt to be used to parse command line options
- Also makes **-a** consistent with our **-o** and **-s** switch naming convention

Harbison & Steele Readings

- Even though no specific readings from **Harbison & Steele** are assigned in the syllabus, everyone should be reading and referring to H&S on a regular basis

- Familiarity with Chapters 1, 2, and 4 through 9 is required and assumed

Programs Accepted by our Grammar

- The grammar is not sufficient to determine if a program is valid
- There are many examples where the grammar accepts invalid programs
- Each section of H&S contains constraints that must be checked to ensure that a valid program exists

Problem Set 2

- **Problem Set 2**
  - Was due Sunday, October 8th, 2023 at midnight Eastern Time

- Problem Set 2 involves building the AST

Problem Set 3

- **Problem Set 3**
  - Was due this past Sunday, October 15th, 2023 at midnight Eastern Time

- Problem Set 3 involves building symbol tables & type data structures

Problem Set 4 (1 of 2)

- **Problem Set 4**
  - Will be due Sunday, October 29th, 2023 at midnight Eastern Time

- Problem Set 4 involves semantic analysis/type checking
  - Creation of type trees
  - Functions written to perform conversions (usual casting conversions, usual assignment conversions, usual unary conversions, usual binary conversions, etc.)
  - Implicit casts are made explicit by adding casts to the AST
  - Each AST node in an expression must be labelled with a correct type tree
  - Each AST node in an expression must also be labelled as either an lvalue/rvalue or an rvalue and whether that type is modifiable or not
  - Errors should be emitted for violations of semantic/type requirements

Problem Set 4 (2 of 2)

- Work first on creating type trees
- Then, starting with simple expressions and building to more complex expressions, add functions for conversions as needed and insert explicit casts and label all AST nodes in expressions with type trees, lvalue vs. rvalue, and modifiability
- As progressing, emit errors for invalid type checks
- Leave the most complex casts for last, *i.e.,* those for ++ and -- and compound assignment operators

The Proctorio Setup Quiz

- *As soon as possible,* take the **Proctorio Setup Quiz** that is available in **Canvas** under **Quizzes**
- Really **carefully read all of the instructions**
  - **The instructions are the same as for the real Midterm Exam**
  - **We are not able to deal with issues caused by not following the instructions!**
- Contact the course staff *as soon as possible* if you have any problems taking the **Proctorio Setup Quiz**

Midterm Exam (1 of 2)

- Our **Midterm Exam begins next week** on October 24th, 2023 at 8:00 PM ET
  - There will be **no class meeting next week on October 24th**, but **section will still be held on October 24th**
    - No topics relevant to the midterm exam will be discussed in that section meeting

- Ask any questions relevant to the midterm exam in section, in class, in office hours, or on Ed before noon ET on the day of the exam

Midterm Exam (2 of 2)

- Exam is **three hours in duration**
  - The exam is designed to take two hours
  - But, the exam is time-consuming – don't worry if you take all three hours
  - The problems are scored in points where a point is weighted to be approximately one minute of exam answer duration
- Earliest starting time is 8:00 PM ET on Tuesday, October 24th, 2023 – **one week from today**
- Latest starting time is 7:59 PM ET on Wednesday, October 25th, 2023
- The exam will be available through **Canvas** under the **Quizzes** tab
  - The exam will be administered under **Proctorio**
  - **Either Google Chrome, Microsoft Edge, Brave, or Opera must be used as the browser** for the exam – only these browsers are acceptable to Proctorio
  - Proctorio requires that a **Proctorio Extension** be installed into your browser
- No books may be used during the exam, but all slides from class will be available through the **Files** menu item in Canvas (a link to the **Files** page of Canvas is in the exam Instructions)
  - Nothing else can be used during the exam: no notes, no electronics other than the computer being used for the exam, no cell phones, no communication
  - But, nothing else is needed!

Midterm Exam Format

- Different question formats
  - Essay
    - Text box in which answer can be typed
  - File Upload
    - Create your answer using: (1) a drawing application on your computer, (2) a drawing app from a web site, or (3) a piece of paper whose scan or photo is submitted.
    - In any of these cases, you will need to ***upload your answer file***. You must ensure that you have uploaded any answer files before the termination of the exam time. **No files can be uploaded after time has expired.**
  - Multiple Choice
    - Choose the correct answer
  - Multiple Answers
    - Select all correct answers

Material on Midterm Exam

- Book readings
  - Aho, Lam, Sethi & Ullman, Chapters 1-6
  - Harbison & Steele
    - Familiarity with Chapters 1, 2, and 4 through 9 is required and assumed
- Material covered in class through – and including – next week's class meeting except for Intermediate Representation (if we get that far)
- Material covered in Problem Sets 1-3
- Discussions on Ed
- Material covered in section is beneficial, and is required

Topics for Midterm Exam (1 of 3)

- C Programming Language
- Flex/Lex
- Regex's
- Context-Free Grammars
- Ambiguity
- Left Recursion
- Left Factoring
- Bison/YACC
- The Grammar for Our Subset-C Language
- NFA, DFA
  - Creating an NFA from a Regex using the McNaughton-Yamada-Thompson Algorithm
  - Conversion of NFA to DFA using Subset Construction

Topics for Midterm Exam (2 of 3)

- Top-Down Parsing
- Recursive Descent Parsing
- Symbol Table Management
- Types in the C Programming Language
- Representing Types in Our Compiler
- Parse Tree, AST
- Name Spaces
- FIRST, FOLLOW, LL(1) Grammars, Predictive Parsing Table M, Table-Driven Predictive Parsing
- Number Representations

Topics for Midterm Exam (3 of 3)

- Type Checking
- C Standard Conversions
- Bottom-Up Parsing
- Shift-Reduce Parsing

Midterm Directions

- Read each question very carefully
  - The question should include all necessary information
- The point values in the exam are equal to the number of minutes that a student who would score highly might take to complete the question
- The exam in lengthy – don’t be concerned if you don’t complete everything
- Initially, spend approximately the number of minutes on each question that the question is worth, then return to the question later to complete them as available time permits

Reviewing Upcoming Dates

- PS3 was already due: Sunday, October 15th, 2023 at midnight ET

- This is the last class meeting before the Midterm Exam
- And, we have no more class meetings before PS4 is due

- Section will meet at 6:45 PM to 7:45 PM ET on Tuesday, October 24th, 2023
- No class meeting next week

- Midterm Exam
  - Earliest start time is 8:00 PM ET on Tuesday, October 24th, 2023
  - Must be started before 8:00 PM ET on Wednesday, October 25th, 2023
  - Must be completed within 3 hours of the time the student starts the exam

- PS4 Due: Sunday, October 29th, 2023 at midnight ET

Issues with Cross-Compiling

- Our compiler runs on the host computer (our cscie95.dce.harvard.edu instance which is **X86_64**)
- Our compiler produces code for our target computer (**32-bit MIPS**)
- So, it's not correct to use information about the host computer to determine minimum and maximum values that are representable on our target computer
  - Therefore, don't use the <limits.h> header file for our target computer

- For example, on our **host computer**, a char has 1 byte, a short int has 2 bytes, an int has 4 bytes, and a **long int has 8 bytes**
- On our **target computer**, a char has 1 byte, a short int has 2 bytes, an int has 4 bytes, and a **long int has 4 bytes**

New Material for this Week (1 of 6)

- Present the **Shift-Reduce Parsing** slides
  - Bottom-Up LR(*k*) Parsing
  - **Shift-Reduce Parsing**
  - Sufficient for all modern computer languages, including C
  - Our students don't need to be able to create an LR-parsing table – just be able to execute a parser using it

New Material for this Week (2 of 6)

- **Numeric Encodings**

New Material for this Week (3 of 6)

- **Character Encodings**

New Material for this Week (4 of 6)

- [Show symbol tables for the code in symbolTables.c]

New Material for this Week (5 of 6)

- [Look at some system header files on cscie95.dce.harvard.edu]
  - They reside in directory /usr/include

  - /usr/include/stdint.h
  - /usr/include/limits.h

New Material for this Week (6 of 6)

- Present the **Intermediate Representation** slides
  - Cover through the **Generating Code for Subtrees** slide

Eighth Class Meeting on 10/31/2023

- Ninth class meeting if you count the midterm exam meeting

- Happy Halloween!

Eighth Class Meeting Agenda

- Questions and Comments
- Sequential Nature of Problem Sets
- Current Problem Set Status
  - PS4
  - PS5
- Midterm Exam
- Errata from Harbison & Steele
- Students Not Yet Working on PS4
- Important Tables in Harbison & Steele
- Cover the **Intermediate Representation** slides
- Cover the **MIPS Instruction Set** slides

Questions and Comments

- Section
- Last week's class or any previous classes
- Problem Sets 0 through 4
- Midterm Exam
- Symbol Table
- Type Tree Creation
- Type Checking
- Labelling All AST *Expression* Nodes with Type Tree, Lvalue vs. Rvalue, and Modifiability
- Making Implicit Casts Now Be Explicit
- Our grammar
- Review of the C Programming Language
- Ed Discussion threads
- Readings
- Anything else

Sequential Nature of Problem Sets

- The Problem Sets are designed to not require modifications to an earlier problem set in order to implement the requirements of a later problem set
  - Problems need to be fixed
  - Required features that were bypassed need to be implemented
  - But, you should be able to take the output of PS*n* without modification and use it to implement PS*n+1*

Problem Set 4 (1 of 3)

- **Problem Set 4**
  - Was due this past Sunday, October 29th, 2023 at midnight Eastern Time

- Problem Set 4 involves semantic analysis/type checking
  - Creation of type trees
  - Functions written to perform conversions (usual casting conversions, usual assignment conversions, usual unary conversions, usual binary conversions, etc.)
  - Implicit casts are made explicit by adding casts to the AST
  - Each AST node in an expression must be labelled with a correct type tree
  - Each AST node in an expression must also be labelled as either an lvalue/rvalue or an rvalue and whether that type is modifiable or not
  - Errors should be emitted for violations of semantic/type requirements

Problem Set 4 (2 of 3)

- Work first on creating type trees
- Then, starting with simple expressions and building to more complex expressions, add functions for conversions as needed and insert explicit casts and label all AST nodes in expressions with type trees, lvalue vs. rvalue, and modifiability
- As progressing, emit errors for invalid type checks
- Leave the most complex casts for last, *i.e.,* those for ++ and -- and compound assignment operators

Problem Set 4 (3 of 3)

- Insertion of casts in PS4
  - Explicit casts are inserted into the AST in PS4 when the type of the result of an operator is not the same as the type when that result is used as an operand of another operator
  - It is possible that more than one cast would be inserted into the AST for a single implicit type conversion
    - For example, one cast might be added as a result of applying the **Usual Unary Conversions** and another added as a result of applying the **Usual Binary Conversions**
    - If this is the case, then multiple casts should be added to the AST

Problem Set 5 (1 of 2)

- Problem Set 5
  - Will be due on Sunday, November 12th, 2023 at midnight Eastern Time
- Present Problem Set 5
- Problem Set 5 involves generating intermediate code (IR)

Problem Set 5 (2 of 2)

- The most important design methodology is to remember to follow the approach where expressions are evaluated to an *lvalue* (when possible) and are evaluated to an *rvalue* only when necessary
  - Expressions are evaluated to an *rvalue* when:
    - The result of an operator is defined to never be an *lvalue*
    - The operand of an operator requires its operand to be an *rvalue*
- Tag each AST node in an expression with
  - The name of the temporary that holds the *lvalue* or *rvalue* of that subexpression
  - Whether that temporary is an *lvalue* or an *rvalue*

Midterm Exam

- We’re still reviewing and grading the midterm exams
- Next week, I hope to present midterm exam statistics and review the answers

Errata from Harbison & Steele

- Link to Harbison's errata is on the class web site
- There is also a link to our own errata from H&S

Students Not Yet Working on PS4

- Students who are not yet working on PS4 may choose the following path
  - If you have not yet started work on PS4, it is possible to delay work on PS4 or, perhaps, to never work on PS4
  - If you are certain that a type-correct program is given to your compiler, then PS4 would not change the AST in any way (except for inserting some casts in a few cases)
  - Therefore, in these cases, we suggest that you skip working on PS4 for now and implement PS5 *before* PS4
  - This gives you a shortcut to being able to run code

- If you haven't already done so, you will have to label every node in an expression in the AST with the type of that node
  - This would normally happen in PS4, but if PS4 is being skipped for now, it is still needed for PS5
  - This is required to know which IR instruction should be generated

Important Tables in Harbison & Steele

- **Table 7-1** on Page 204
  - Nonarray expressions that can be lvalues
- **Table 7-2** on Page 204
  - Operators requiring lvalue operands

Model for Intermediate Representation

- Two different storage areas in which data can be stored
  - Temporaries/registers
  - Memory

- Each location (byte) in memory has an address
- A temporary/register has a name, but not an address

- The load*Type* and store*Type* IR instructions are the only way to copy data between temporaries/registers and memory

New Material for this Week (1 of 2)

- Present the **Intermediate Representation** slides
  - Continue from the **Deriving an rvalue from an lvalue (1 of 3)** slide

New Material for this Week (2 of 2)

- Present the **MIPS Instruction Set** slides
  - Cover through the **Add Instruction** slide

Ninth Class Meeting on 11/7/2023

- Tenth class meeting if you count the midterm exam meeting

Ninth Class Meeting Agenda

- Questions and Comments
- Current Problem Set Status
  - PS4
  - PS5
  - Present PS6
- Students Not Yet Working on PS4
- Midterm Exam
  - Statistics
  - Review of Solutions
- Cover the **MIPS Instruction Set** slides

Questions and Comments

- Section
- Last week's class or any previous classes
- Problem Sets 0 through 5
- Symbol Table
- Type Tree Creation
- Type Checking
- Labelling All AST *Expression* Nodes with Type Tree, Lvalue vs. Rvalue, and Modifiability
- Making Implicit Casts Now Be Explicit
- Our grammar
- Review of the C Programming Language
- Ed Discussion threads
- Readings
- Anything else

Problem Set 4

- **Problem Set 4**
  - Was due on Sunday, October 29th, 2023 at midnight Eastern Time

- Problem Set 4 involves semantic analysis/type checking
  - Creation of type trees
  - Functions written to perform conversions (usual casting conversions, usual assignment conversions, usual unary conversions, usual binary conversions, etc.)
  - Implicit casts are made explicit by adding casts to the AST
  - Each AST node in an expression must be labelled with a correct type tree
  - Each AST node in an expression must also be labelled as either an lvalue/rvalue or an rvalue and whether that type is modifiable or not
  - Errors should be emitted for violations of semantic/type requirements

Problem Set 5 (1 of 2)

- **Problem Set 5**
  - Will be due this coming Sunday, November 12th, 2023 at midnight Eastern Time

- Problem Set 5 involves generating intermediate code (IR)

Problem Set 5 (2 of 2)

- The most important design methodology is to remember to follow the approach where expressions are evaluated to an *lvalue* (when possible) and are evaluated to an *rvalue* only when necessary
  - Expressions are evaluated to an *rvalue* when:
    - The result of an operator is defined to never be an *lvalue*
    - The operand of an operator requires its operand to be an *rvalue*
- Tag each AST node in an expression with
  - The name of the temporary that holds the *lvalue* or *rvalue* of that subexpression
  - Whether that temporary is an *lvalue* or an *rvalue*

Problem Set 6

- **Problem Set 6**
  - Will be due on Sunday, November 26th, 2023 at midnight Eastern Time

- Present Problem Set 6

- Problem Set 6 involves generating MIPS assembly code suitable for SPIM

Students Not Yet Working on PS4

- Students who are not yet working on PS4 may choose the following path
  - If you have not yet started work on PS4, it is possible to delay work on PS4 or, perhaps, to never work on PS4
  - If you are certain that a type-correct program is given to your compiler, then PS4 would not change the AST in any way (except for inserting some casts in a few cases)
  - Therefore, in these cases, we suggest that you skip working on PS4 for now and implement PS5 *before* PS4
  - This gives you a shortcut to being able to run code

- If you haven't already done so, you will have to label every node in an expression in the AST with the type of that node
  - This would normally happen in PS4, but if PS4 is being skipped for now, it is still needed for PS5
  - This is required to know which IR instruction should be generated

Midterm Exam Statistics

Name	Midterm 1	Midterm 2	Midterm 3	Midterm 4	Midterm 5	Midterm 6	Midterm 7	Midterm 8	Midterm 9	Midterm 10	Midterm 11	Midterm Total
	Variable Type	Invalid IDs	Array Convrsn	Seq Points	Reg Exp	NFA	DFA	Lex	YACC	Left Recursion Elimination	Generate M	
Max Possible	2	1	1	1	5	10	15	15	30	15	25	120
Count	9	9	9	9	9	9	9	9	9	9	9	9
Minimum	0	0	0	0	2	2	4	0	13	4	0	47
Maximum	2	1	1	1	5	10	15	14	25	15	22	107
Average	1.333333333	0.666666667	0.777777778	0.555555556	3.777777778	7.666666667	12.22222222	10.11111111	21.55555556	11.33333333	10.22222222	80.22222222
St. Dev.	1	0.353553391	0.440958552	0.527046277	1.092906421	2.449489743	3.597838857	4.675586713	3.609401305	4.769696007	8.212456663	16.17697266

Midterm Exam

- Go over the answers to the midterm exam questions

New Material for this Week

- Present the **MIPS Instruction Set** slides
  - Continue from the **Instruction Set Architecture (ISA) Descriptive Information** slide
    - Cover through the **Branch I-Type Instructions** slide

Tenth Class Meeting on 11/14/2023

- Eleventh class meeting if you count the midterm exam meeting

Agenda

- Questions and Comments
- Current Problem Set Status
  - Late Problem Set Point Deductions
  - Problem Sets 5 & 6 and Upcoming Due Dates
- Compiler Implementation Reminders and Notes
  - Ignore All References to Delay Slot
  - Generate Code to Deal with Integral Value Added to a Pointer
  - IR Code Generated for Casts to Pointer Types from Integral Types and vice versa
  - Prefixes for Static Identifiers and for All Labels
  - Signed and Unsigned IR and MIPS Instructions
  - Generating Branch and Jump Instructions
- New Material
  - Continue with the **MIPS Instruction Set** slides
  - Cover the **MIPS Assembly Language** slides
  - Present the MIPS assembly code examples and run them under SPIM
  - Cover the **Run-time Environment** slides
  - Present an overview of the factorial code examples

Questions and Comments

- MIPS Instruction Set and Assembly Language
- Midterm exam
- Problem Set 3
  - Type Trees
  - Symbol Tables
- Problem Set 4
  - Type Checking
- Problem Set 5
  - Intermediate Representation (IR) Generation
- Problem Set 6
  - Generating MIPS assembly code suitable for SPIM
- Section or lecture
- Ed
- Readings
- Anything else

Late Problem Set Point Deductions

- There will ***not*** be any points deducted for late problem sets this semester

Problem Set 5

- Problem Set 5
  - Was due this past Sunday, November 12th, 2023 at midnight Eastern Time

- Problem Set 5 involves generating intermediate code (IR)

Problem Set 6

- Problem Set 6
  - Will be due on Sunday, November 26th, 2023 at midnight Eastern Time

- Problem Set 6 involves generating MIPS assembly code suitable for SPIM

Ignore All References to Delay Slot

- We are using a simplified model of the MIPS computer that does not implement the delay slot
- Therefore, you should ignore all references to “delay slot” in my MIPS Instruction Set slides
  - Also ignore the MULT description text that talks about “if either of the two preceding instructions…”

Generate Code to Deal with Integral Value Added to a Pointer

- Generate IR code to appropriately scale an integral expression that is added to a pointer
  - See **Intermediate Representation** slides, **IR Code for Addition of an Integer to a Pointer** slide 25
  - See **Intermediate Representation** slides, **IR Code for Subscript Operator with Arrays** slide 26

- When adding an integral expression to a pointer, the signedness of the IR instruction generated should be the same as the signedness of the integral expression (*i.e.,* when adding a signed integral value use the signed add IR instruction and when adding an unsigned integral value, use the unsigned add IR instruction)

IR Code Generated for Casts to Pointer Types from Integral Types and vice versa

- This note concerns the IR code to be generated for a cast **to** any pointer type **from** any integral type and vice versa
- The representation of a pointer in IR/MIPS is a single word (four bytes)
- Casts between pointer types and integral types require generation of casting IR instructions *only when the sizes of the integral type and of a pointer are different*
- ①Consider: int i, *pi; pi = (int *)i; i = (int)pi;
  - No cast IR is required for these casting conversions because they change the type for our type calculus, but not the representation of the int or pointer
- ②Consider: char c, *pc; pc = (char *)c; c = (char)pc;
  - Cast IRs are required for these casting conversions because in addition to changing the type for our type calculus, they change the representation of the char (one byte in size) to a pointer (four bytes in size) or vice versa
- ③Similarly for: short int si, *psi; psi = (short int *)si; si = (short int)psi;
- ④And, similarly for: char c; int i; short int *psi; psi = (short int *)c; i = (int)psi;
- **Keep in mind that when the sizes of the integral type and a pointer are different as in cases ② through ④ above, that the integral values/pointers are probably not useful or usable because of widening of the integral value/truncation of the pointers**

Prefixes for Static Identifiers and for All Labels

- Don’t generate a prefix for the **`main`** function

- _Global_*fileScopeIdentifier*
- _UserLabel_*functionName*_*userLabel*
- _GeneratedLabel_*integer*
- _StringLabel_*integer*

Signed and Unsigned IR and MIPS Instructions (1 of 2)

- Even though we will be generating both signed and unsigned IR instructions, we will *never* generate any MIPS instructions that can cause exceptions
  - Because MIPS computers (and all modern computers) use two's-complement representation for signed integers, the instructions for addition and subtraction are the same for signed and for unsigned integral types
  - Therefore, never generate ADD or SUB MIPS instructions
  - Instead use ADDU or SUBU instructions
- Do generate MULT or MULTU, DIV or DIVU as appropriate for signed or unsigned operations, respectively
- For right shift operations, generate SRL or SRLV for unsigned operands and generate SRA or SRAV for signed operands

Signed and Unsigned IR and MIPS Instructions (2 of 2)

- Never generate an ADDI MIPS instruction because it can cause an exception
- ADDI and ADDIU always sign-extend their immediate operands
- ANDI, ORI, and XORI always zero-extend their immediate operands
- Both SLTI and SLTIU sign-extend their immediate operands, but SLTI compares as signed integers and SLTIU compares as unsigned integers
- All of the BLEZ, BGTZ, BLTZ, and BGEZ instructions treat their operand as a signed integer
- Never generate a BLTZAL or BGEZAL instruction because calling a function is more complicated and requires more instructions (as we'll see later today)

Generating Branch and Jump Instructions

- Branch instructions are limited in the offset from the current PC value
  - Don’t worry about this when you are first generating code
  - Use Branch instructions for all operations that change flow of control within a function
  - The assembler will emit an error if the branch distance to the target is too far large
- Always use the Jump And Link (JAL) instruction to call a function
  - It will overwrite the link register ($ra ($31)) with the return address
  - Therefore, use `JR $ra` to return to the caller

New Material for this Week (1 of 5)

- Continue with the **MIPS Instruction Set** slides
  - Continue after the **Branch I-Type Instructions** slide

New Material for this Week (2 of 5)

- Present the **MIPS Assembly Language** slides

New Material for this Week (3 of 5)

- Present the MIPS assembly code examples and run them under SPIM or QtSpim
  - printint.s
  - printstring.s
  - readstring.s
  - count.s
  - count2.s
  - count3.s

New Material for this Week (4 of 5)

- Present the **Run-time Environment** slides

New Material for this Week (5 of 5)

- Note that this example uses the $t registers before the $s registers which is *not* what you'll be doing in your generated MIPS code
  - I'm doing this to later show how to optimize saving and restoring of the $t registers

- Present an overview of the factorial code examples
  - factorial - No strcpy.c
  - factorial - No strcpy.ir
  - factorial - No strcpy.s

Eleventh Class Meeting on 11/21/2023

- Twelfth class meeting if you count the midterm exam meeting

Agenda

- Questions and Comments
- Administrivia
- Current Problem Set Status
  - Late Problem Set Point Deductions
  - Problem Sets 5 & 6 and Upcoming Due Dates
  - Final Project
- Compiler Implementation Reminders and Notes
  - See last week's Class Agenda slides for these reminders and notes
- New Material
  - Present a few additional MIPS assembly code examples and run them under SPIM
  - Review the **Run-time Environment** slides
  - Cover the **Determining Stack Offsets** slides
  - Present the factorial code examples

Questions and Comments

- MIPS Instruction Set
- MIPS Assembly Language
- Run-time Environment
- Midterm exam
- Problem Set 3
  - Type Trees
  - Symbol Tables
- Problem Set 4
  - Type Checking
- Problem Set 5
  - Intermediate Representation (IR) Generation
- Problem Set 6
  - Generating MIPS assembly code suitable for SPIM
- Section or lecture
- Ed
- Readings
- Anything else

Administrivia

- Office hours normally held on Thursday this week will be moved to Wednesday 7:00 to 8:00 PM ET
  - This is due to the US Thanksgiving holiday on Thursday

Late Problem Set Point Deductions

- There will ***not*** be any points deducted for late problem sets this semester

Problem Set 5

- Problem Set 5
  - Was due on Sunday, November 12th, 2023 at midnight Eastern Time

- Problem Set 5 involves generating intermediate code (IR)

Problem Set 6

- Problem Set 6
  - Will be due on Sunday, November 26th, 2023 at midnight Eastern Time

- Problem Set 6 involves generating MIPS assembly code suitable for SPIM

Final Project (1 of 2)

- Present the Final Project description

- Final Project
  - ***Pre-recorded audio and video presentation*** to class on **Tuesday, December 19th, 2023 beginning at 6:45 PM ET** – four weeks from today
    - Class meeting will begin promptly at **6:45 PM ET** in our usual Zoom ***lecture*** room
    - Slots for each presentation
    - Each presentation is followed by a **five minute *live Q & A session*** over Zoom
  - By **4 PM ET on Tuesday, December 19th, 2023**, we need to have received the URL of your **ten minute** audio and video presentation
  - **Final code is due by Midnight ET on Thursday night, December 21st, 2023**

Final Project (2 of 2)

- The Final Project involves code optimization
  - All optimizations must be controlled by a command line switch
    - -O 0 means perform no optimization (default)
    - -O 1 means perform minimal optimization
    - -O 2 means perform slightly more optimization
    - etc.
  - Almost all optimization will be performed at the IR level
    - You are welcome to add IR instructions, if necessary
    - Produce more efficient code
    - You should have a number of C programs that you'll use in your presentation
      - Carefully analyze the straight-forward assembler code that you generate without any optimizations
      - Optimize the assembler code for these benchmark programs and show the performance improvement
  - Some other optimizations
    - Resetting register usage at the beginning of each statement
    - Generating more efficient code to save and restore $s and $t registers
      - That is, not just saving all $s and $t registers
    - Better register allocation (graph coloring?)

New Material for this Week (1 of 4)

- Present the MIPS assembly code examples and run them under SPIM or QtSpim
  - squares.s
  - storedints.s

New Material for this Week (2 of 4)

- Review the **Run-time Environment** slides

New Material for this Week (3 of 4)

- Present the **Determining Stack Offsets** slides

New Material for this Week (4 of 4)

- Note that these examples all use the $t registers before the $s registers which is *not* what you'll be doing in your generated MIPS code
  - I'm doing this to show how to optimize saving and restoring of the $t registers

- Present the factorial code examples
  - factorial - No strcpy.c
  - factorial - No strcpy.ir
  - factorial - No strcpy.s

  - factorial.c
  - factorial.ir
  - factorial - Simple Code.s

  - factorial.s # This has optimized saving and restoring of $t registers

Twelfth Class Meeting on 11/28/2023

- Thirteenth class meeting if you count the midterm exam meeting

Agenda

- Questions and Comments
- Administrivia
  - Late Problem Set Point Deductions
  - Problem Set 6 and Upcoming Due Dates
  - Final Project
    - Register Allocation
- New Material
  - Present the **Optimization** slides
  - Show how **factorialTailRecursive.c** is optimized into **factorialTailRecursiveOptimized.c**
  - Present the **Next-Use, Liveness & Register Allocation** slides
  - Present **Chordal Graphs** paper
  - Preview **Casting Conversions** slides

Questions and Comments

- IR Generation
- MIPS Instruction Set and Assembly Language
- Run-time Environment
  - Stack frame format and contents
  - Determining stack frame offsets
- Final Project
- All Problem Sets
- Section or lecture
  - factorial.c
  - factorial.ir
  - factorial.s
- Ed
- Readings
- Anything else

Late Problem Set Point Deductions

- There will ***not*** be any points deducted for late problem sets this semester

Problem Set 6

- Problem Set 6
  - Was due this past Sunday, November 26th, 2023 at midnight Eastern Time

- Problem Set 6 involves generating MIPS assembly code suitable for SPIM

Final Project (1 of 2)

- Just ***three weeks until*** **Final Project presentations** during section and class time
- Final Project
  - ***Pre-recorded ten minute (timing enforced)*** ***audio and video presentation*** to class on **Tuesday, December 19th, 2023 beginning at 6:45 PM ET** – three weeks from today
    - Class meeting will begin promptly at **6:45 PM ET** in our usual classroom and in the Zoom ***lecture*** room
    - Ten minute slots for each presentation
    - Each presentation is followed by a **five minute *live Q & A session*** both in our usual classroom and over Zoom
  - By **4 PM ET on Tuesday, December 19th, 2023**, we need to have received the URL of your **ten minute** audio and video presentation
  - **Final code is due by Midnight ET on Thursday night, December 21st, 2023**

Final Project (2 of 2)

- The Final Project involves code optimization
  - All optimizations must be controlled by a command line switch
    - -O 0 means perform no optimization (default)
    - -O 1 means perform minimal optimization
    - -O 2 means perform slightly more optimization
    - etc.
  - Almost all optimization will be performed at the IR level
    - You are welcome to add IR instructions, if necessary
    - Produce more efficient code
    - You should have a number of C programs that you'll use in your presentation
      - Carefully analyze the straight-forward assembler code that you generate without any optimizations
      - Optimize the assembler code for these benchmark programs and show the performance improvement
  - Some other optimizations
    - Resetting register usage at the beginning of each statement
    - Generating more efficient code to save and restore $s and $t registers
      - That is, not just saving all $s and $t registers
    - Better register allocation (graph coloring?)

How to Apply Register Allocation

- When IR temporary number is reset before each statement…
  - Reduce use of temporaries/registers for code generated for a statement
- When no function is called in a statement…
  - It’s fine to use both $s and $t registers because the $t registers will not be overwritten
- When at least one function is called in a statement…
  - Try not to use any $t registers
  - This would allow the function to be called without saving any $t registers
- These rules are an easy way to determine how many registers may be used in graph coloring

More Advanced Application of Register Allocation

- Don’t reset the temporary number before each statement in IR generation
- Use the specific computer architecture to determine the number of registers that are available
  - Map IR temporaries to hardware registers during MIPS code generation
  - Determine basic blocks
  - Assign IR temporaries to hardware registers within each basic block
  - Follow the same guidelines as on the previous slide concerning function calls

Even More Advanced Register Allocation

- Perform the following optimizations within basic blocks
  - When the address of a user variable is placed into a temporary using the IR addressOf opcode, if the variable could not have moved in memory, reuse the old temporary and remove the redundant addressOf
  - When performing a loadWord, loadHalf, or loadByte, determine if a temporary has already been loaded with the appropriate value and if the value could not have been altered in memory
    - Requires determining if a store might have altered the memory location between the earlier load and the later load
    - If the location could not have been altered, reuse the old temporary and remove the redundant loadWord, loadHalf, or loadByte
    - Beware of possible accesses via pointer dereferencing

New Material for this Week (1 of 5)

- Present the **Optimization** slides

New Material for this Week (2 of 5)

- Optimization
  - Show how **factorialTailRecursive.c** is optimized into **factorialTailRecursiveOptimized.c**

New Material for this Week (3 of 5)

- Present the **Next-Use, Liveness & Register Allocation** slides

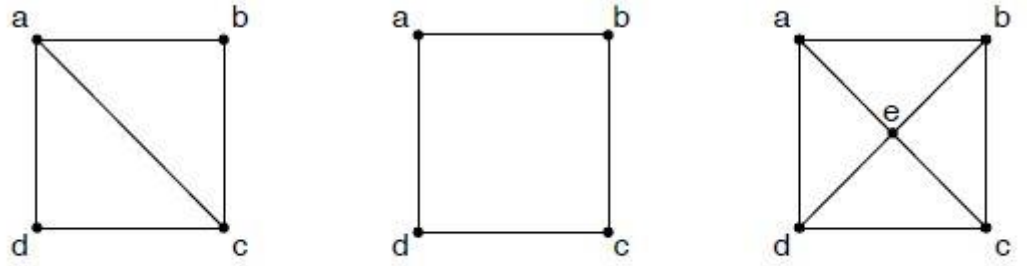
Chordal Graphs (4a of 5)

- See paper: *Register Allocation via Coloring of Chordal Graphs* by Fernando Magno Quintão Pereira and Jens Palsberg, UCLA Computer Science Department

- A graph is *chordal* if every cycle with four or more edges has a chord, that is, an edge which is not part of the cycle but which connects two vertices on the cycle. (Chordal graphs are also known as *triangulated*, *rigid-circuit*, *monotone transitive*, and *perfect elimination* graphs.)

Chordal Graphs (4b of 5)

- (a) is a Chordal Graph
- (b) & (c) are Non-Chordal Graphs



Chordal Graphs (4c of 5)

- The authors observed that the interference graphs of real-life programs tend to be chordal graphs
- The paper reports that, for example, 95% of the methods in the Java 1.5 library have chordal interference graphs when compiled with the JoeQ compiler
- Problems such as *minimum coloring*, *maximum clique*, *maximum independent set* and *minimum covering by cliques*, which are NP-complete in general, can be solved in polynomial time for chordal graphs [Fanica Gavril. Algorithms for minimum coloring, maximum clique, minimum covering by cliques, and maximum independent set of a chordal graph. SICOMP, 1(2):180-187, 1972]
- This paper shows that, in particular, optimal coloring of a chordal graph G = (V, E) can be done in O(|E| + |V|) time

New Material for this Week (5 of 5)

- Preview **Casting Conversions** slides

Thirteenth Class Meeting on 12/5/2023

- Fourteenth class meeting if you count the midterm exam meeting

Agenda

- Questions and Comments
- Administrivia
  - Remaining Class Meetings
  - Late Problem Set Point Deductions
  - Problem Sets
  - Final Project
  - Final Presentation Slots
- Passing and Accessing Parameters to Functions
- New Material
  - Present **Casting Conversions** slides
  - Additional SYSCALLs
  - Present the **Dependencies, Instruction Scheduling, Optimization, and Parallelism** slides
    - Present Overview of the **Pipelining** and **Caching** slides from CSCI E-93
    - Present Overview of the **Jack Dennis model of Data Flow Computation**

Questions and Comments

- Final Project/Optimizations

- IR Generation
- Generating MIPS assembly code suitable for SPIM
- Run-time Environment
  - Stack frame format and contents
  - Determining stack frame offsets
- Next-Use, Liveness, Graph Coloring & Register Allocation
- All Problem Sets
- Section or lecture
  - Factorial examples
- Ed
- Readings
- Anything else

Remaining Class Meetings

- Today is our second to last class meeting in which material will be presented
- Next week on December 12th will be our last class meeting in which material will be presented
- Two weeks from today on December 19th we will meet for student compiler final project presentations

Late Problem Set Point Deductions

- There will ***not*** be any points deducted for late problem sets this semester

Problem Sets

- All Problem Sets have already been due

Final Project (1 of 2)

- Just ***two weeks until*** **Final Project presentations** during section and class time
- Final Project
  - ***Pre-recorded ten minute (timing enforced)*** ***audio and video presentation*** to class on **Tuesday, December 19th, 2023 beginning at 6:45 PM ET** – two weeks from today
    - Class meeting will begin promptly at **6:45 PM ET** in our usual classroom and in the Zoom ***lecture*** room
    - Ten minute slots for each presentation
    - Each presentation is followed by a **five minute *live Q & A session*** both in our usual classroom and over Zoom
    - **Refer to https://cscie95.dce.harvard.edu/fall2023/FinalProject.txt**
  - By **4 PM ET on Tuesday, December 19th, 2023**, we need to have received the URL of your **ten minute** audio and video presentation
  - **Final code is due by Midnight ET on Thursday night, December 21st, 2023**

Final Project (2 of 2)

- The Final Project involves code optimization
  - All optimizations must be controlled by a command line switch
    - -O 0 means perform no optimization (default)
    - -O 1 means perform minimal optimization
    - -O 2 means perform slightly more optimization
    - etc.
  - Almost all optimization will be performed at the IR level
    - You are welcome to add IR instructions, if necessary
    - Produce more efficient code
    - You should have a number of C programs that you'll use in your presentation
      - Carefully analyze the straight-forward assembler code that you generate without any optimizations
      - Optimize the assembler code for these benchmark programs and show the performance improvement
  - Some other optimizations
    - Resetting register usage at the beginning of each statement
    - Generating more efficient code to save and restore $s and $t registers
      - That is, not just saving all $s and $t registers
    - Better register allocation (graph coloring?)

Final Presentation Slots (All Eastern Time)

- Slot 1, 6:45-7:00 PM: Keenan
- Slot 2, 7:00-7:15 PM: Lucas
- Slot 3, 7:15-7:30 PM: Abreeza
- Slot 4, 7:30-7:45 PM:
- Slot 5, 7:45-8:00 PM: Adi
- Slot 6, 8:00-8:15 PM: Yaz
- Slot 7, 8:15-8:30 PM:
- Slot 8, 8:30-8:45 PM:
- Slot 9, 8:45-9:00 PM:

Passing and Accessing Parameters to Functions

- Each function's entry code will save any actual parameter values into slots in the callee's stack frame
  - That is, on function entry, any values passed in registers $a0 through $a3 as the first four actual parameters will be saved into the appropriate entries in the callee's stack frame
  - Within the body of the function, the values of the first four formal parameters are accessed in memory using the offsets in the stack frame where $a0 through $a3 were saved, as appropriate
  - As shown in the **Stack Frame Format** slide in my **Run-time Environment** slides, registers $a0 through $a3 are saved in the **General Register Save Area**

New Material for this Week (1 of 3)

- Present **Casting Conversions** slides

New Material for this Week (2 of 3)

- Additional SYSCALLs

Service	Code in $v0	Arguments	Result
print integer	1	$a0 = integer to print	
print float	2	$f12 = float to print	
print double	3	$f12 = double to print	
print string	4	$a0 = address of null-terminated string to print	
read integer	5		$v0 contains integer read
read float	6		$f0 contains float read
read double	7		$f0 contains double read
read string	8	$a0 = address of input buffer $a1 = maximum number of characters to read	Follows semantics of UNIX fgets
sbrk (allocate heap memory)	9	$a0 = number of bytes to allocate	$v0 contains address of allocated memory
exit (terminate execution)	10		
print character	11	$a0 = character to print	Character should be in low-order byte
read character	12		$v0 contains character read
open file	13	$a0 = address of null-terminated string containing filename $a1 = flags $a2 = mode	$v0 contains file descriptor (negative if error); flags is 0 for read, 1 for write; mode can be set to 0
read from file	14	$a0 = file descriptor $a1 = address of input buffer $a2 = maximum number of characters to read	$v0 contains number of characters read (0 if end-of-file, negative if error)
write to file	15	$a0 = file descriptor $a1 = address of output buffer $a2 = number of characters to write	$v0 contains number of characters written (negative if error)
close file	16	$a0 = file descriptor	
exit2 (terminate with value)	17	$a0 = termination result	

New Material for this Week (3 of 3)

- Present the **Dependencies, Instruction Scheduling, Optimization, and Parallelism** slides through the **Sequential Array Accesses (1 of 3)** slide

Fourteenth Class Meeting on 12/12/2023

- Fifteenth class meeting if you count the midterm exam meeting

Agenda

- Questions and Comments
- Administrivia
  - Remaining Class Meetings
  - Late Problem Set Point Deductions
  - Problem Sets
  - Papers Available on the Class Website
  - Final Project
  - Final Presentation Schedule
  - Course Evaluation
  - Celebratory Dinner After Final Presentations
  - Upcoming Classes
  - Ski Trip between the fall and spring semesters
- Wrinkles about Passing Parameters to Functions
- New Material
  - Present **Mark Ford's Compilers** video
  - Continue with the **Dependencies, Instruction Scheduling, Optimization, and Parallelism** slides
  - Present Overview of the **Jack Dennis model of Data Flow Computation**
  - Present the **C* Language** for programming Massively-Parallel Computers

Questions and Comments

- Final Project/Optimizations

- IR Generation
- Generating MIPS assembly code suitable for SPIM
- Run-time Environment
  - Stack frame format and contents
  - Determining stack frame offsets
- Next-Use, Liveness, Graph Coloring & Register Allocation
- Casting Conversions
- Additional SYSCALLs
- Basic Blocks and Dependency Analysis
- Pipelining
- Caching
- All Problem Sets
- Section or lecture
  - Factorial examples
- Ed
- Readings
- Anything else

Remaining Class Meetings

- Today is our last class meeting in which material will be presented
- We will meet next week on Tuesday, December 19th, 2023 for student compiler final project presentations

Late Problem Set Point Deductions

- There will ***not*** be any points deducted for late problem sets this semester

Problem Sets

- All Problem Sets have already been due

Papers Available on the Class Website

- On the class website under Software and Course Documents On-Line → Papers Used in Class, the following papers are available for your enjoyment
  - Register Allocation and Spilling via Graph Coloring by Gregory J. Chaitin, IBM T. J. Watson Research Center, ACM Proceedings of the 1982 SIGPLAN symposium on compiler construction, 1982, p.98-105: Also in Best of PLDI 1979-1999
  - Register Allocation via Coloring of Chordal Graphs by Fernando Magno Quintão Pereira and Jens Palsberg, UCLA Computer Science Department, APLAS'05: Proceedings of the Third Asian conference on Programming Languages and Systems

Final Project: Overview

- Just ***one week until*** **Final Project presentations** during section and class time
- Final Project
  - ***Pre-recorded ten minute (timing enforced)*** ***audio and video presentation*** to class on **Tuesday, December 19th, 2023 beginning at 6:45 PM ET** – one week from today
    - Class meeting will begin promptly at **6:45 PM ET** in our usual classroom and in the Zoom ***lecture*** room
    - Ten minute slots for each presentation
    - Each presentation is followed by a **five minute** ***live Q & A session*** both in our usual classroom and over Zoom
    - **Refer to https://cscie95.dce.harvard.edu/fall2023/FinalProject.txt**
  - By **4 PM ET on Tuesday, December 19th, 2023**, we need to have received the URL of your **ten minute** audio and video presentation
  - **Final code is due by Midnight ET on Thursday night, December 21st, 2023**
- We may have visitors present for the presentations
- Your term project grade includes the video presentation

Final Project: Optimizations

- The Final Project involves code optimization
  - All optimizations must be controlled by a command line switch
    - -O 0 means perform no optimization (default)
    - -O 1 means perform minimal optimization
    - -O 2 means perform slightly more optimization
    - etc.
  - Almost all optimization will be performed at the IR level
    - You are welcome to add IR instructions, if necessary
    - Produce more efficient code
    - You should have a number of C programs that you'll use in your presentation
      - Carefully analyze the straight-forward assembler code that you generate without any optimizations
      - Optimize the assembler code for these benchmark programs and show the performance improvement
  - Some other optimizations
    - Resetting register usage at the beginning of each statement
    - Generating more efficient code to save and restore $s and $t registers
      - That is, not just saving all $s and $t registers
    - Better register allocation (graph coloring?)

Final Project: Presentation

- Included in your presentation should be slides/video that show:
  - functionality of all phases of your compiler using command line switches
  - your internal representations (tokens, AST, IR, MIPS code)
  - intermediate output from your compiler (symbol table, IR)
  - interesting design, testing, and code generation aspects of your compiler
  - examples of noteworthy programs being compiled and executed under SPIM
  - which language features are not yet implemented
  - optimizations that are implemented
  - demo of several C programs being compiled and executed both before and after optimization and the improvements in space and time that are seen

Final Project: Logistics

- All students will participate in the question and answer portion

Final Project: Code & Document Submission

- **By Midnight ET on Thursday night, December 21st, 2023**, each class member should submit (with git tag term-project) their Final Project which will include:
  - the slides used during your presentation
  - C code for your compiler
  - testing code that you used
  - any documentation that you created for your compiler

Final Presentation Schedule (All Eastern Time)

- Slot 1, 6:45-7:00 PM: Keenan
- Slot 2, 7:00-7:15 PM: Lucas
- Slot 3, 7:15-7:30 PM: Abreeza
- Slot 4, 7:30-7:45 PM: Alejandro
- Slot 5, 7:45-8:00 PM: Adi
- Slot 6, 8:00-8:15 PM: Yaz
- Slot 7, 8:15-8:30 PM: Rodrigo
- Slot 8, 8:30-8:45 PM: Yohei
- Slot 9, 8:45-9:00 PM: Marie

Course Evaluation

- Please fill out the course evaluation form that will soon be available to you

- Course Evaluation is available on-line from December 14, 2023 to January 2, 2024

Dinner Get-together After Final Presentations

- Meet after the final class meeting for a celebratory dinner get-together in Harvard Square
- Significant others are invited to join us

Upcoming Classes

- Spring 2024 – CSCI E-92: Principles of Operating Systems
  - Concepts, issues, and implementation of an operating system
  - Culmination is designing and implementing an OS for an NXP ARM processor including shell, memory allocation, device drivers, device-independent I/O, FAT32 file system, and multiprocessing

- Fall 2024 – CSCI E-93: Computer Architecture
  - Study of how computers function
  - Course begins with principles of electricity, digital logic, data path for a computer, control logic for a computer, fluency with VHDL, and instruction set design
  - Culmination is designing and implementing a new, individually-created and implemented block diagram, instruction set, and processor from digital logic on an Altera FPGA
  - Advanced topics: Caching, Pipelining, VM, MMU, Vector CPU, VLIW, MPP

Ski Trip between the fall and spring semesters

- We plan to have a **class ski trip on Sunday, January 21st, 2024** to Killington Ski Area in Vermont
  - How many of you are interested in participating in this event?
  - I'll contact class members via e-mail with specifics

- Day outing
  - Meet in the morning near Boston or at the mountain
  - Depending on ability, ski together
  - All meet for lunch
  - Dinner together in the evening on the way home

Wrinkles about Passing Parameters to Functions (1 of 4)

- Even if you have passing actual parameters to called functions working, there are a couple more wrinkles
- If you have a nested call to a function, you may not produce correct code because of two issues:
  - Correctly numbering the parameters
  - Not overwriting the $a registers
- Imagine the following C statement (where i, j, k, and m are all ints):

```
f(i, g(j, k), m);
```

- What IR would you produce?

```
(addressOf, r0, i)          # r0 -> i
(loadWord, r1, r0)          # r1 <- i
(parameter, 0, r1)          # First (#0) parameter to f is i
(addressOf, r2, j)          # r2 -> j
(loadWord, r3, r2)          # r3 <- j
(parameter, 0, r3)          # First (#0) parameter to g is j
(addressOf, r4, k)          # r4 -> k
(loadWord, r5, r4)          # r5 <- k
(parameter, 1, r5)          # Second (#1) parameter to g is k
(call, _Global_g)           # Call g
(resultWord, r6)            # r6 <- return value from g
(parameter, 1, r6)          # Second (#1) parameter to f is the return value from g
(addressOf, r7, m)          # r7 -> m
(loadWord, r8, r7)          # r8 <- m
(parameter, 2, r8)          # Third (#2) parameter to f is m
(call, _Global_f)           # Call f
```

Wrinkles about Passing Parameters to Functions (2 of 4)

Produced IR

```
(addressOf, r0, i)
(loadWord, r1, r0)
(parameter, 0, r1)
(addressOf, r2, j)
(loadWord, r3, r2)
(parameter, 0, r3)
(addressOf, r4, k)
(loadWord, r5, r4)
(parameter, 1, r5)
(call, _Global_g)
(resultWord, r6)
(parameter, 1, r6)
(addressOf, r7, m)
(loadWord, r8, r7)
(parameter, 2, r8)
(call, _Global_f)
```

MIPS Code

```
la      $s0, i_offset($fp)
lw      $s1, 0($s0)
or      $a0, $s1, $0
la      $s2, j _offset($fp)
lw      $s3, 0($s2)
or      $a0, $s3, $0
la      $s4, k _offset($fp)
lw      $s5, 0($s4)
or      $a1, $s5, $0
jal     _Global_g
or      $s6, $v0, $0
or      $a1, $s6, $0
la      $s7, m _offset($fp)
lw      $s8, 0($s7)
or      $a2, $s8, $0
jal     _Global_f
```

Wrinkles about Passing Parameters to Functions (3 of 4)

- We can resolve this problem by issuing the parameter IRs immediately before the call IR

```
(addressOf, r0, i)  # r0 -> i
(loadWord, r1, r0)  # r1 <- i
(addressOf, r2, j)  # r2 -> j
(loadWord, r3, r2)  # r3 <- j
(addressOf, r4, k)  # r4 -> k
(loadWord, r5, r4)  # r5 <- k
(parameter, 0, r3)  # First (#0) parameter to g is j
(parameter, 1, r5)  # Second (#1) parameter to g is k
(call, _Global_g)   # Call g
(resultWord, r6)    # r6 <- return value from g
(addressOf, r7, m)  # r7 -> m
(loadWord, r8, r7)  # r8 <- m
(parameter, 0, r1)  # First (#0) parameter to f is i
(parameter, 1, r6)  # Second (#1) parameter to f is the return value from g
(parameter, 2, r8)  # Third (#2) parameter to f is m
(call, _Global_f)   # Call f
```

Wrinkles about Passing Parameters to Functions (4 of 4)

Produced IR

```
(addressOf, r0, i)
(loadWord, r1, r0)
(addressOf, r2, j)
(loadWord, r3, r2)
(addressOf, r4, k)
(loadWord, r5, r4)
(parameter, 0, r3)
(parameter, 1, r5)
(call, _Global_g)
(resultWord, r6)
(addressOf, r7, m)
(loadWord, r8, r7)
(parameter, 0, r1)
(parameter, 1, r6)
(parameter, 2, r8)
(call, _Global_f)
```

MIPS Code

```
la      $s0, i_offset($fp)
lw      $s1, 0($s0)
la      $s2, j_offset($fp)
lw      $s3, 0($s2)
la      $s4, k_offset($fp)
lw      $s5, 0($s4)
or      $a0, $s3, $0
or      $a1, $s5, $0
jal     _Global_g
or      $s6, $v0, $0
la      $s7, m_offset($fp)
lw      $s8, 0($s7)
or      $a0, $s1, $0
or      $a1, $s6, $0
or      $a2, $s8, $0
jal     _Global_f
```

New Material for this Week (1 of 4)

- Presentation of our TA, **Mark Ford's, Compilers** video (yes, *compilers plural*)

New Material for this Week (2 of 4)

- Present the **Dependencies, Instruction Scheduling, Optimization, and Parallelism** slides continuing with the **Sequential Array Accesses (1 of 3)** slide

New Material for this Week (3 of 4)

- Present the **Jack Dennis model of Data Flow Computation**

New Material for this Week (4 of 4)

- Present the **C* Language** for programming Massively-Parallel Computers (think supercomputer GPU)