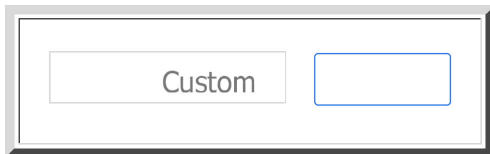
CS255 Syllabus & Progress

Material covered are displayed in blue.

- Intro to CS255: click here

Search the lecture notes:



1. Review of Computer Architecture and Intro to Program Organization
    - **Logical View** of a Computer System: click here
    - Structure of computer main memory: click here
    - Accessing the computer memory: click here
    - Program organization (where different parts of the program is put in memory): click here
    - The program translation process: click here

Before we can explain how the computer works - i.e., how it manipulates data, we must understand how information are stored inside the computer...

2. How information are stored in the computer
    - Fundamental data used in the computer (these values need to be stored)
        - Numbers (int, float, double)
        - Text (char)
        - Computer instructions
        - (Boolean values "true" is represented as the number 1 and "false" as number 0)
    - Storing **numeric values** inside the computer:
        - Numeric values: click here
        - Positional value representation: click here
        - Representing unsigned integers: click here
        - Arithmetic with binary numbers: click here
    - Other number systems:
        - Base-5 numbers: click here
        - Octal numbers: click here
        - Hexadecimal numbers: click here
    - Representing **Signed** integers: click here
        - 10's complement number **encoding**: click here
        - 2's complement number **encoding**: click here

        - "Excess-n": an alternate **binary encoding** for signed values: click here

    - Fixed point numbers: click here
    - Floating point numbers
        - Floating point representation: click here
        - Floating point round off error: Float.java

    Homework 1 re-enforces number system concepts: click here

    ---

    - A binary number information sheet: click here

3. Representing **alphanumerical data** inside the computer:
    - ASCII code: click here
    - Communicating **textual data** between humans and computers: click here
    - Communicating **non-textual information** to a computer:
        - Communicating **Boolean values** between humans and computers: click here
        - Communicating **numerical values** between humans and computers:
            - A **trivial solution**: click here
            - Working with ASCII codes: click here
            - Some **Java** facts: click here
            - Communicating **numerical values** between humans and computers: click here

        ---

        Homework 2 re-enforces string <-> binary conversion: click here

        ---

    - Storing computer instruction inside the computer:
        - Instruction categories:
            - 0 operand
            - 1 operand (unary operation, e.g., negate)
            - 2 operands (binary operation, e.g., add)
        - Instruction formats:
            - Variable length (Intel, Motorola M68000)
            - Fixed length (SPARC, MIPS)
        - M68000 Instruction encoding: click here
        - SPARC Instruction encoding: click here
    - What's that in the memory ? click here

---

4. **Intro to assembler programming**
    - Intro: click here
    - From high level programming language to machine executable code: click here
    - General CPU architecture: click here
    - CPU operation: instruction execution cycle: click here

---

5. **Intro to M68000 assembler programming**

    - Why assembler programming ? click here
    - M68000 CPU architecture: click here
    - Operands in M68000:
        - Intro: click here
        - Data Register operands: click here
        - Address Register operands: click here
        - Memory operands: click here
        - Operand data type and assembler instructions: click here
    - Intro to assembler programming: click here
    - M68000 Assembler directives: click here
    - M68000 Assembler Instruction: click here

    - **M68000 instruction nmenomic codes and their meaning:** click here
    - **M68000 instruction nmenomic codes and the allowed addressing modes:** click here
    - All M68000 instructions discussed in CS255 (work in progress): click here

    ---

6. **The Assignment Statement:**
    - A simple example: C = A + B click here
    - **M68000 addressing modes** (**tough topic**...)
        - Immediate: click here
        - Absolute or Direct: click here

        Homework 3 introduces the Egtapi programming environment, and re-enforces defining and using simple variables with Assembler Directives and the Immediate & Direct Modes: click here

        This project is very easy to do, but make sure you assemble and run the program to get familiar with Egtapi, or else, your future encounter with Egtapi with be very unpleasant if you have to struggle with bugs in the assembler program *and* finding out on how to operate Egtapi...

        - Address register indirect (with displacement): click here
        - Accessing data structures using the indirect modes:
            - Accessing arrays: click here
            - How is a linked list stored in memory: click here
            - Accesing data in linked list: click here
        - Address register indirect with **index** and displacement: click here

- **Arithmetic operations:**
  - The add, adda, sub, suba instructions: click here
  - The negate instruction: click here
  - The **muls** and **divs** instructions (simple multiply and divide): click here

- **Mixing `int`, `short`, `byte` operands:**
  - Converting between integer of **different sizes**: click here
  - **Assignment statement** using **different sizes**: click here
  - **Arithmetic expression** using **different sizes**: click here
  - Mixed type operations with **array variables**: click here

  - Using the **ext** instruction on the results of the **muls** and **divs** instructions:
    - Multiply: click here
    - Divide: click here

  - A large example accessing data structures (includes linked list): click here

- What about these "complicated" functions - like **sin(x)**: click here

---

Homework 4 re-enforces using array variables and size conversion with M68000 assembler instructions: click here

---

Now you know what the computer actually does when it executes an assignment statement: how it evaluates an expression and assigns the result to a variable in a high level programming language

7. **The Selection (if, if-else) Statements:**
   - The Compare `(cmp)` and Branch instructions of M68000: click here
   - Selection statements in assembler:
     - Simple if: click here
     - Simple if-else: click here
     - Compound condition using `and`: click here
     - Compound condition using `or`: click here
     - Nested If-statements: click here

8. **Repetition statements in assembler:**
   - While-statment: click here
   - For-statment: click here
   - Do-statment: do.s.
   - Traversing linked lists: click here

Homework 5 re-enforces the use of compare and branching in writing your first assembler program containing a while loop and two if-statements: click here

Here is an additional practice to re-enforces linked list access and manipulation (and the WHILE loop construct): click here (will be given only if there is sufficient time)

The midterm test will cover the material upto this point and will be scheduled soon.

Now you know what the computer actually does when it runs a program. The only thing that you still do not know is what happens in a function call (there is still a lot to learn...).

Programming project pj2 re-enforces loop and if-statements further with a sorting algorithm... click here

---

9. **Subroutines (a.k.a. methods, procedures, functions):**
   - Prelude to subroutines: implementing a stack click here
   - Prelude to subroutines: try to use **BRA** to implement subroutines: click here
   - M68000 instructions for subroutine invocation and return click here
   - Passing parameters and return values: click here
   - Different ways to pass parameters to a function/subroutine:
     - Pass-by-value (a.k.a. Call-by-value): click here
     - Pass-by-reference: (a.k.a. Call-by-reference): click here
     - Important note: click here
   - **Local variables (and parameter variables):**
     - **Intro** to Subroutine with local variables: click here
     - Review: **Lifetime** of parameter variables and local variables: click here
     - How **modern programming languages** store **parameter** and **local** variables:
       - Modern languages: click here
       - using the **stack** to **pass parameters** and **store local variables**: click here
       - using a ***frame pointer*** to ***access*** **parameters** and **local variables**: click here
   - **Recursion:**
     - Reminder on parameter variables and local variable:
       - ***parameter variables*** and ***local variables*** are **created** when a function is invoked
       - ***parameter variables*** and ***local variables*** are **destroyed** when a function returns
       - We have learned how to **implement** this behavior using the **system stack**
     - **Writing recursive functions in assembler code:**
       1. always pass parameters on the stack
       2. always use the stack to store local variables
     - First recursion in assembler, the classical factorial: click here
     - Another classic recursive problem -- Fibonacci numbers: click here

---

I inserted some material here to help you understand **how** to **program** using **recursion** --- not part of CS255 curriculum:

- **Review CS170:** recursion is a **divide and conquer** technique to solve problems: click here
- **Review CS170:** Tower of Hanoi click here
- A **tougher** example of **divide and conquer** using **recursion:** click here (Not part of CS255 material)

- Tower of Hanoi in M68000 assembler code: click here (skipped for brevity - read through it yourself)

Programming project **hw7** re-enforces recursion: click here

---

- **Working with linked lists:**
  - Remember how linked lists are stored inside the computer: click here
  - Insert at start of the linked list: click here
  - Insert a new element at the tail of the list:
    - **Iterative algorithm** to insert at tail of a list: click here --- skipped (just loop and if, you can read it)
    - **Recursive algorithm** to insert at tail of a list:
      - My CS171 material on the algorithm: click here
      - M68000 program: click here
  - Insert into an ordered list: as hw8

Programming project hw8 re-enforces linked lists: click here

If you understand everything so far, you now know exactly what is going on when a computer executes a program. You can apply what you learn and write assembler programs in **any** assembler language, e.g., Intel Pentium, IBM PowerChip, DEC Alpha, SunMicrosystem SPARC... To program with a new chip, all you need to learn are:

- its architecture (registers),
- the addressing modes
- the instruction mnemonic codes and what they do.

I have SPARC and Intel material after the C lecture notes to demonstrate this process.

---

10. **Introduction to the (System Programming Language) C for Java programmers**

- ***Introduction*** to **C** programming: click here
- Things that are **identical** in **Java** and **C**: click here
- Introduction to the ***C Pre-processor:***
  - Intro: click here
  - Macros (`#define`): click here
  - Including files (`#include`): click here
  - **Conditional** compilation: click here
  - A **trick** to prevent a file being included ***multiple*** **times**: click here
- C's data types and variables:
  - **C's data types:**
    - Overview: click here
    - Operations involving **same** and ***different*** data types: click here

    - Conversion rules in C: click here
  - The **const** modifier: click here

- Basic input/output:
  - **Printing** values of the built-in data types: click here
  - **Reading in** values of the built-in data types: click here

---

- **Operators**:
  - ***Arithmetic Operators:*** click here (+, -, *, /)
  - ***Bit-wise-Operators:***
    - The **bit-wise** operators: (&, |, ^, ~) click here
    - **How to** use the **bit-wise** operators: click here
    - Denoting values in base 2, 8 and 16: click here
    - Bit manipulation functions - clear a bit, set a bit, flip a bit and test a bit: click here
  - ***Shift-Operators:*** click here (>>, <<)
    - Intro: click here
    - Set bit at position n: click here
    - Clear bit at position n: click here
    - Flip bit at position n: click here
    - Test bit at position n: click here
  - ***Assignment Operators:*** click here (+=, -=, *=, /=, &=, |=, ~=, >>=, <<=)
  - ***Increment/decrement Operators:*** click here (++, --)
  - Boolean data type and operations:
    - The **Boolean *data type*** in **C**: click here
    - **Compare and Logic Operators:** click here (<, <=, >, >=, ==, !=, &&, ||, !)
  - The ***conditional operator* ?:** click here
  - The ***comma operator* ,** click here
  - The **sizeof** operator click here
  - ***Priority*** and ***Associativity*** of the **C operators** click here

---

- **Statements**:
  - ***Assignment Statement:*** click here
  - ***Selection Statements (if, if-else, switch):*** click here
  - ***Loop Statements (while, for, do-while):*** click here

- **Arrays:**
  - **Defining** and **using *one-dimensional*** array variables: click here
  - Review: **static** and **dynamic** arrays: click here

- **Array bound** checks: click here
- **Defining** and **using** ***two-dimensional*** array variables: click here
- "***Bit***" **arrays:** click here

---

11. **Functions (methods):**
    - Intro to ***C Functions:*** click here

    - **Single file** C program: ***Declaring*** **functions:**
        - Preliminary to function declaration:
            - Parameters/return values in a function call: click here
            - **Correctness requirement** on function call with **mismatched** (1) parameter type and (2) return value type: click here
            - **Automatic conversion rules** for function calls: click here
        - **Important differences** between the C and Java compilers:
            - One file vs. multi-file searching: click here
            - **One-pass** vs. **two-pass** compiler: click here
            - **Implicit** function prototyping in **C**: click here
        - Function **declaration**:
            - Function prototype --- ***declaring*** a function: click here
            - Removing the **Implicit** function prototyping in **C**: click here
            - The difference between **defining** and **declaring** a function in **C**: click here

    ---

    - ***Multi-files*** **C programs**: the importance of ***declaring*** **functions**:
        - The ***simplest*** way to compile a **multiple** files C program: click here
        - The ***correct*** way to compile a **multiple** files C program: click here
        - The **importance** of **function declaration** in multi-file C programs: click here
        - Declaring function using **header files**: click here

    ---

    - The ***make*** (UNIX) utility: (SKIP if time constrainted)
        - Specifying dependency between files --- **makefiles**: click here
        - The **make** utility: click here
        - Structure of **makefiles**: click here
        - **Inference rules** used in makefiles: click here
        - **Macros** in makefiles: click here
        - Miscellaneous makefile stuff (line continuation, comment, ...): click here

    ---

    - The ***Parameter passing*** **mechanism** of ***C*****:** click here

---

Hand out **C** Project 1: click here

---

12. **Intermediate topics: things that look similar in *C* and *Java* but work differently**

   - **Variables in *C*:**
     - ***Kinds*** of variables in **C**: click here
     - **Brief review:** ***Life time*** and ***Scope*** of the variables: click here
     - ***Life time*** of variables in ***C***: click here
     - **Scope** of variables in ***C***:
       - **Scoping rules** for: (1) global, (2) `static` global and (3) `static` local variables: click here
       - Declaring **global** variables in **C**: click here
       - ***C Pre-processor*** **trick** to define and declare **global variables**: click here
       - **Scoping rules** for ***local*** and ***parameter*** variables in ***C***: click here ***(pay attention !)***
   - **More** on **arrays**: declaring arrays and array as parameters of functions
     - ***Declaring*** **array** variables: click here
     - **Functions** with ***array*** **parameters**: click here
       - Declaring ***functions*** that have **array parameters**: click here
   - **Debugging** a **C** program:
     - Using the ***gdb*** **debugger**: click here

---

13. ***Advanced*** **datatypes**

   - **Reference** data type and reference (pointer) variables
     - ***Reference (pointer) data type and variables***: click here
     - The ***reference*** **(&)** operator click here
     - Storing the result of ***reference*** **(&)** operator in reference data typed variables: click here
     - Using ***reference*** **variables** --- the ***dereference*** **(*)** operator: click here

     ---

     - **How to** read **C**'s syntax for **variable definition**:
       - How to read "`datatype *`": click here
       - How to read "`const`": click here

     - **Reference variables** as ***parameters*** in a function (**pass-by-reference**): click here
     - Functions **returning** a ***reference*** : click here
     - The **(type *)** casting operator: click here
   - **User-defined types:**
     - ***Structures:***
       - Defining and using **struct** type and variables: click here
       - Declaring **struct** variables: click here
       - **struct** parameter variables: click here
       - Function that returns a **struct**: click here

        - Size of a **struct** "object": click here
    - ***Unions:*** click here
    - ***Enumeration* types:** click here
    - ***Bit fields* structures:** click here

    - ***Pointer*** to user-defined type (just add *): click here
    - The short-hand operator **->**: click here
    - Passing **user-defined typed variables "by *reference*":** click here

    - **C's typing mechanism:** by **(type)** ***name*** **equivalence**: click here
    - **`typedef`**: defining a ***new name*** for a type (***not*** defining a new type): click here

- How to create and maintain **linked lists** in C:
    - **Review** of linked lists and **how to** define linked list elements in **C**: click here
    - **Introduction** to linked lists in **C**: click here
    - **Caveat:** free deleted list elements !!! click here
    - Inserting and deleting at the **head** of a list:
        - Insert at head of list: click here
        - Delete at head of list: click here
    - **Recursion:** Inserting and deleting at the **tail** of a list: click here

---

Hand out **C** Project 2: click here

- **Pointer arithmetic** and **Arrays**
    - ***Pointer arithmetic***: click here
    - Accessing **arrays** with ***pointers*** : click here

    - Using **pointer arithmetic** to implement ***Dynamic arrays***: click here
    - The **[ ]** operator: click here
    - Using the **[ ]** operator to implement ***Dynamic arrays***: click here
    - The ***equivalance*** of **pointers** and **arrays** in **C**: click here

    - The ++ and **--** operators used in pointer arithmetic: click here

---

- We have cover all operators !!. **C's operators** and their precedence: click here

---

- ***Strings*** in **C** --- **arrays** of **`char`:** click here
- Some ***Strings*** **functions** in **C Standard library** click here
- **Array of *Strings*** --- command line arguments: click here

---

14. **Miscellaneous topics in C**

- Special kinds of variables in C:
    - `register` variables: click here

- Function parameters:

- Passing a ***function*** as a parameter to a function: click here

---

Hand out Homework 7 to re-inforce C++ programming click here

That's all, Folks !

---

15. **Additional Enrichment Material: Intel 80x86 programming (*not covered* in CS255)**

   - A free Intel Macro Assembler (if you want to write assembler on your PC): RIGHT click here and SAVE AS
   - An online documentation on Assembler programming with MASM: click here
   - Intel Assembler Code Table: click here
   - The Intel 80x86 CPU architecture: click here
   - The Intel 80x86 directives: click here
   - The Intel 80x86 Addressing modes: click here
   - Arithmetic operations: click here
   - Compare and Branch: click here
   - Subroutine call and return: click here

   - The Art of Assembler Language Programming book: click here

     Check out chapter 6

Rough notes on Intel Assemler programming:

   - Intel registers: click here
   - Intel registers: click here
   - Reserving space for variables: click here
   - Addressing modes (MOV): click here
   - Arithmetic: click here
   - CMP and Jump: click here
   - Stack, procedure call/return: click here

Other resources:

   - Matloff GNU assembler programming tutorial: click here
   - Sourceforge GNU assembler programming tutorial: click here
   - Free Intel Macro Assembler (if you want to write assembler on your PC): RIGHT click here and SAVE AS
   - Intel Assembler Programming book - this book is like the CS255 course using Intel Assembler code: click here

- Step by step Windows programming tutorial shows you how to write window programs, menus, call backs, etc: click here
- The Art of Assembly Programming (a very good online book, but the 32 bit version uses a non-standard assembler called "HLA" - High Level Assemble language) click here