CMPE-013/L

Computer Systems and "C" Programming

Max Lichtenstein

Roadmap

- ✓ Hello and Welcome
- ✓ CMPE13 Overview
- ✓ What is C?
- ✓ How does C work?
- ✓ Basic elements of a C program
- ✓ Key Concepts of CE13
- ✓ Lab01 / git demo (?)

Hello and Welcome

About: Me

- From Denver, CO
- BA from University of Colorado Boulder in Physics
- Working on a PhD in Computer Engineering
  - Working in Gabe Elkaim's Autonomous Systems Lab
  - Researching energy saving and scavenging techniques in animal tracking collars
  - Teaching CE13, apparently

TA

- Pavlo Vlastos
- Undergrad degree from UCSC
- Also working on a PhD in Computer Engineering
  - Also working in Gabe Elkaim's Autonomous Systems Lab
  - Friendly guy

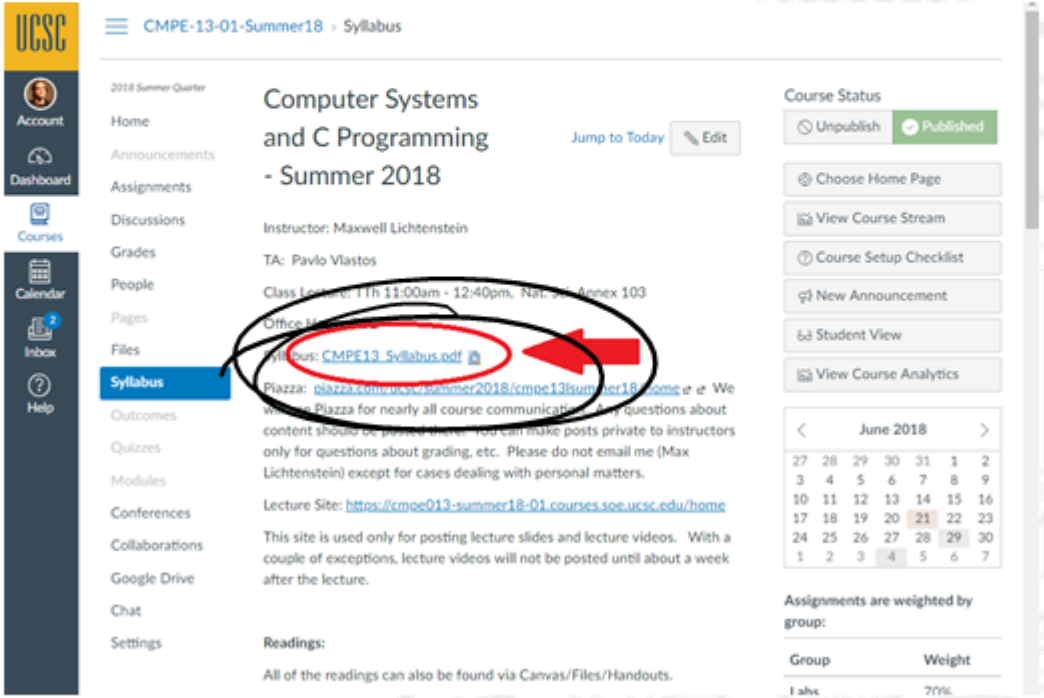
CMPE13 Overview

Class Resources

- CANVAS ✓
- Piazza ✓ All course communication
- GitLab ✓
- Auxiliary Website ✓
  - Storage for lecture videos and slides

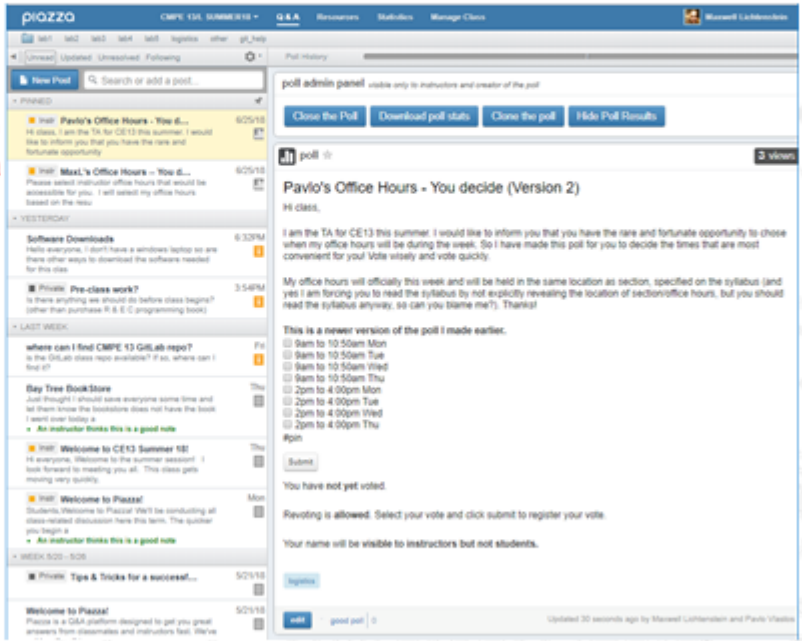
CMPE-013/L Syllabus

- Please read the syllabus!

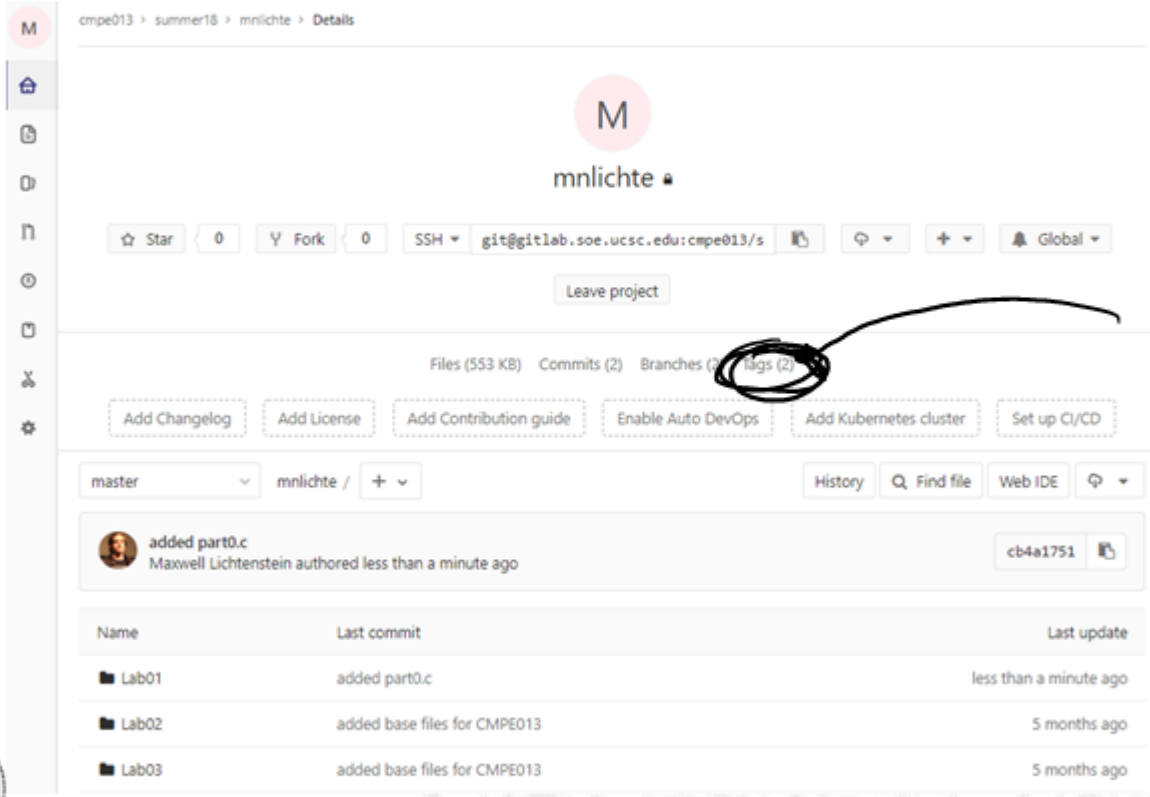


CMPE-013/L Piazza

- Ask!
- Please vote for office hours! (will select by Friday)

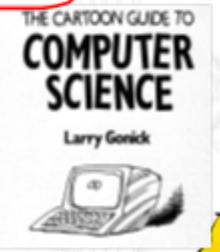


GitLab


cmpe013 > summer18 > mnlichte > Details
M
mnlichte
Star 0 Fork 0 SSH git@gitlab.soe.ucsc.edu:cmpe013/s Global
Leave project
Files (553 KB) Commits (2) Branches (2) Tags (2)
Add Changelog Add License Add Contribution guide Enable Auto DevOps Add Kubernetes cluster Set up CI/CD
master mnlichte / History Find file Web IDE
added part0.c
Maxwell Lichtenstein authored less than a minute ago
cb4a1751
Name Last commit Last update
Lab01 added part0.c less than a minute ago
Lab02 added base files for CMPE013 5 months ago
Lab03 added base files for CMPE013 5 months ago

CMPE-013/L Books

- **[K&R]**: "The C Programming Language, 2nd Edition" by Kernighan and Ritchie, Prentice-Hall, 1988, ISBN-10: 0131103628.
- **[Notes]**: "Notes to accompany K&R," by Steve Summit available on the class website and at: http://www.eskimo.com/~scs/cclass/krnotes/top.html
- "The Cartoon Guide to Computer Science" by Larry Gonick, Barnes and Noble Books, 1983. http://www.soe.ucsc.edu/classes/cmpe013/Spring11/Gonick/



Academic Honesty

- Cheating is presenting someone else's work as your own
- All code turned in will be run against a code-checker
- Anyone caught cheating will immediately fail the class and the lab, and be reported to their college
- Copying each other's code is ***never*** acceptable.

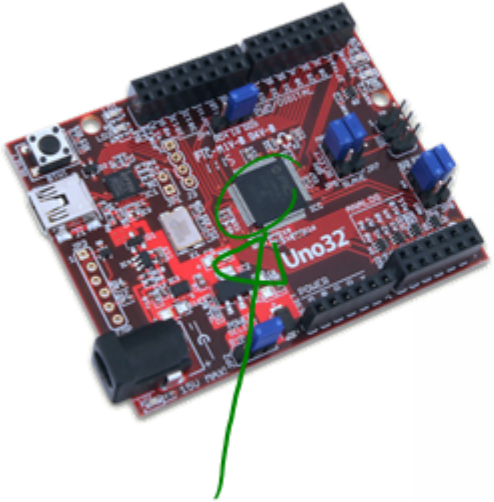
- Don't do it—not worth it.

CMPE-013/L Lab Work

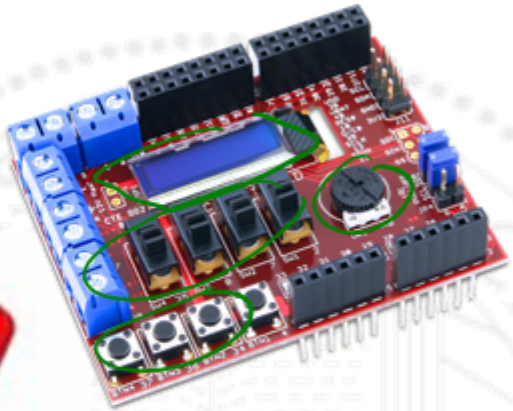
- All programs we are using can be loaded onto your own laptop for use at any time—they are all free.
- We're using microchip's MPLABX IDE and XC32 compiler PLIB
- Running on a microcontroller development board by Digilent, the Uno32 or µ32 (essentially the same board) ✓
- You can buy the this hardware directly from Microchip if you want to after the class is over

Lab Kit

U32 / Uno32



Basic I/O Shield



PICkit3

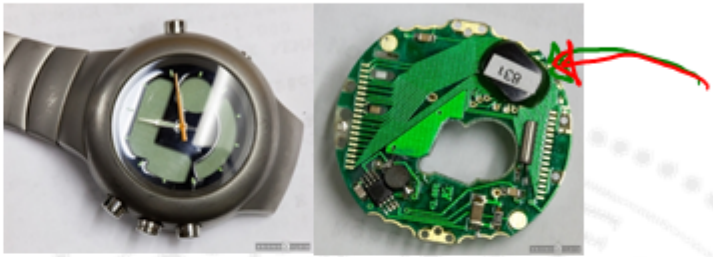
What Is C?

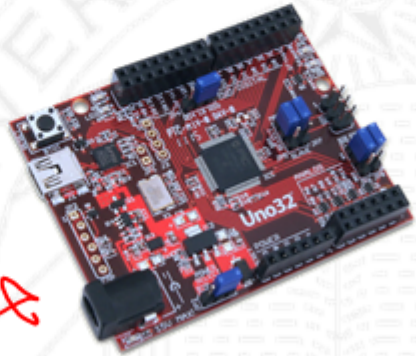
What is a Computer?

- Computer
  - A *computer* is a machine that manipulates data based on a list of instructions called *program*.
  - A computer consists of hardware and software.

- Computer Hardware
  - *Computer hardware* is the physical part of a computer.
  - A computer consists of central processing unit (CPU), memory , and input and output devices.
    - A CPU consists of control unit (CU), arithmetic and logic unit (ALU), and registers.

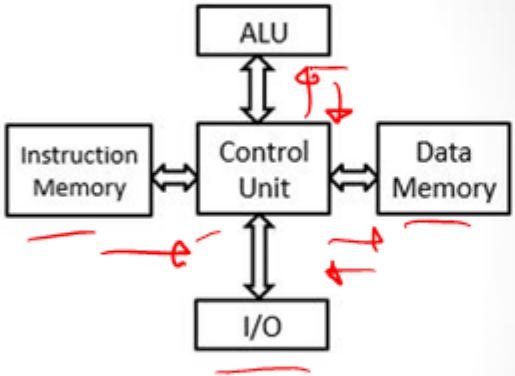
A Few Examples of Computer Hardware



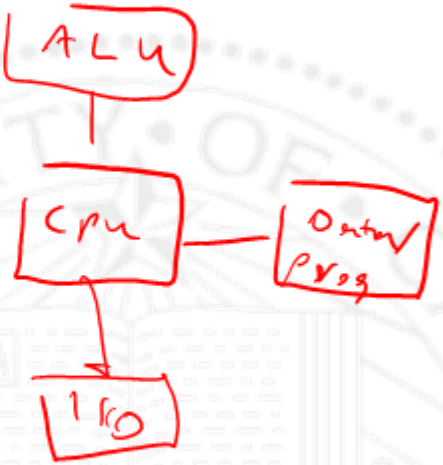


PDP-11 (1972), u32 (2017), Xronos Smart Watch (2013)

Inside a computer:

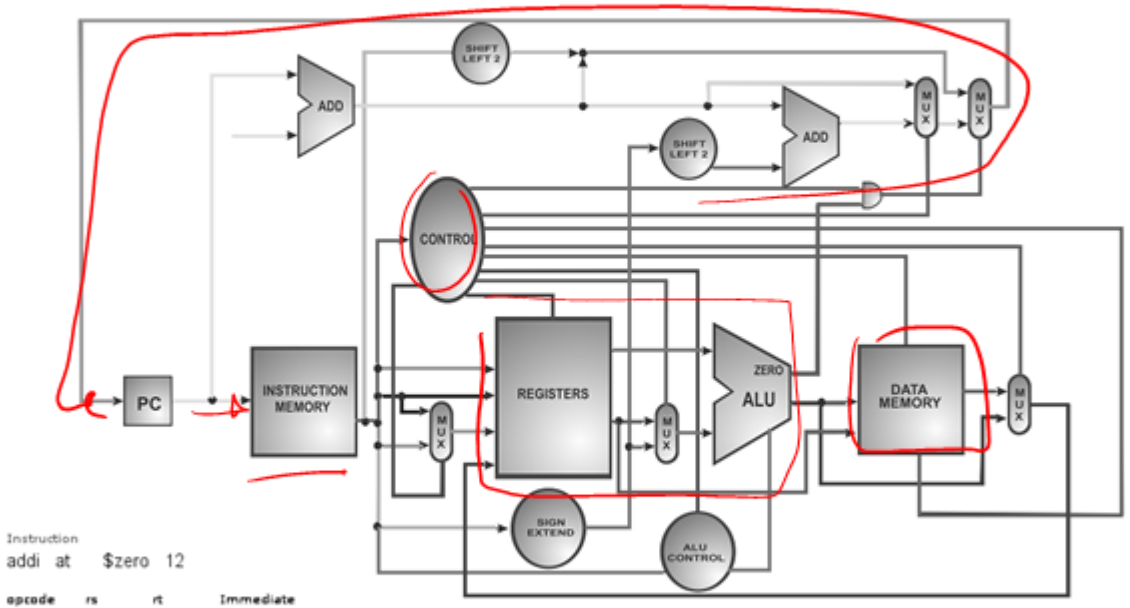


Harvard Model

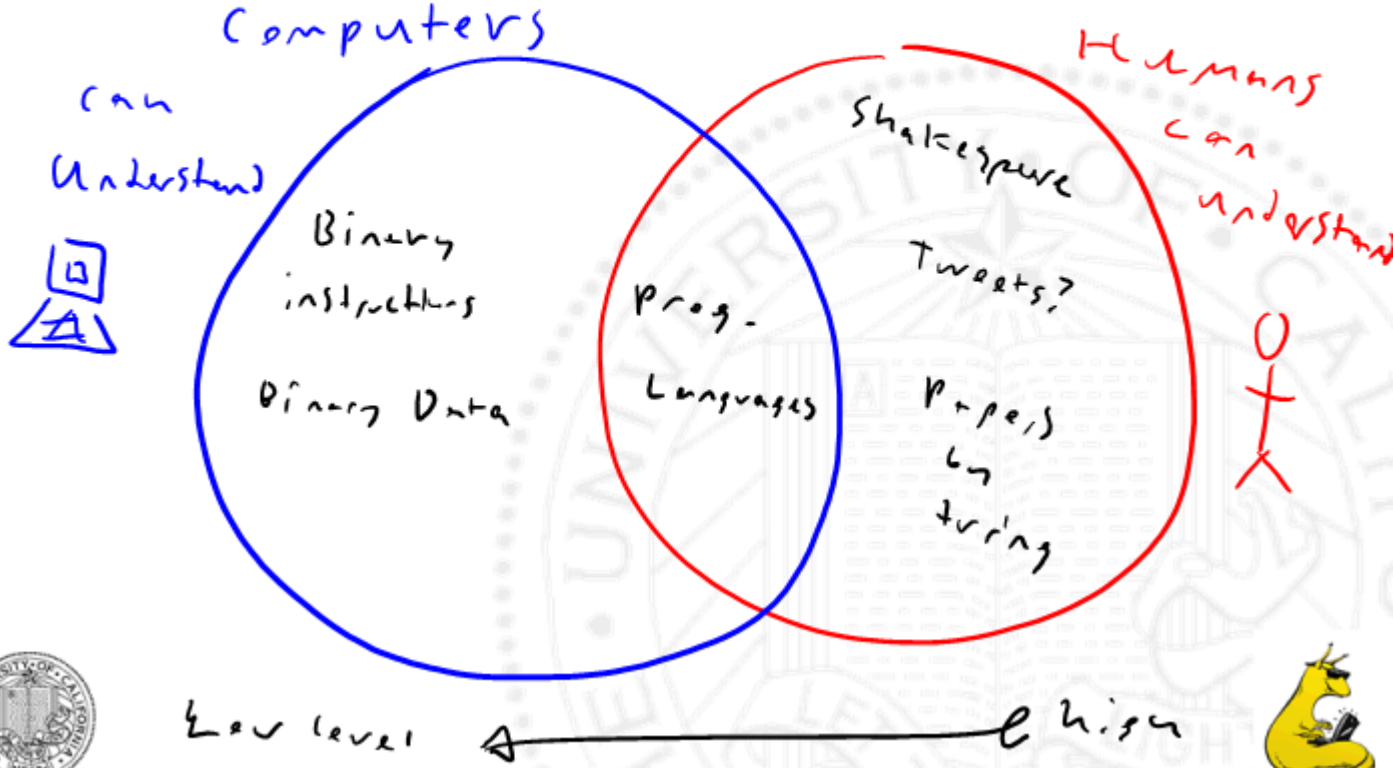


Max Lichtenstein

Inside a computer:



What is a programming language?



Computer Languages

High-level vs Low-level languages

1. Very low level: Machine code

   A sequence of 0's and 1's

2. Assembly language

   Using meaningful symbols to represent machine code.

   Example: add hl,de

Assembler: Assembly code → machine code

Disassembler: machine code → assembly code

Computer Languages

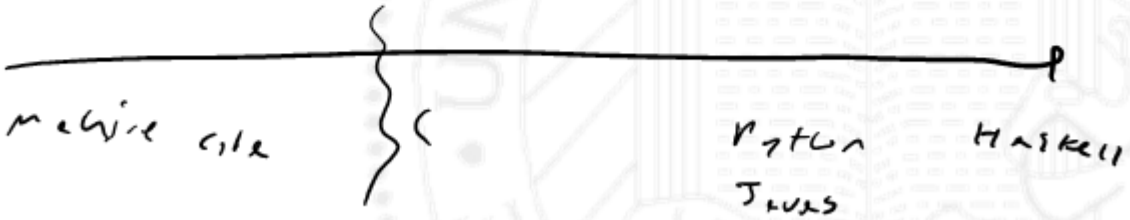
3. High-level languages

Similar to everyday English and use mathematical notations

(processed by compilers or interpreters)

Example of a C statement:

```
a = a + 8;
```



Programming Languages

Many differences including:

1. Abstraction type
2. Compiled vs. interpreted
3. Memory management
4. Type system

The C Programming Language

- Procedural
- Compiled
- Manual memory management
- Statically typed
- Small
- Low overhead

Comparison of High-Level Language with Machine Code and Assembly Code

The memory addresses, machine code, and assembly code corresponding to a C statement *a = a + 8* for the Rabbit 3000 8-bit microprocessor.

Memory address	Machine code	Assembly code	C
0X1EA1	000100010000100000000000	ld de,0x0008	
0X1EA4	1100010000000000	ld hl,(sp+0)	a = a + 8
0X1EA6	00011001	add hl,de	
0X1EA7	1101010000000000	ld (sp+0),hl	

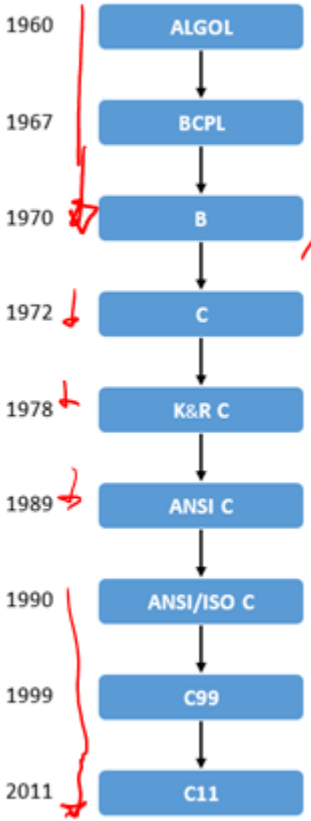
Max Lichtenstein

Reasons to learn C

1. C is one of foundations for modern information technology and computer science
2. C is one of the most common programming languages in the world
3. Embedded software is almost exclusively written in C
4. Unix, macOS, windows, Linux are mostly written in C
5. The parsers, interpreters, and virtual machines for other languages are mostly written in C
6. Learning C teaches you how computers work
7. C makes it easy to learn other languages, because all other modern languages borrowed heavily from C
8. C code often runs much faster than more abstract languages
9. You're enrolled in CE13

Reasons NOT to use C

1. It's often counter-intuitive
2. You have to think about pointers, underlying data structures, and hardware
3. It's kind of ugly
4. There are lots of different implementations, and they don't all behave the same way
5. It's not necessarily *the* most common language…
6. In practice, not very portable



History of C

- C
  - Designed in 1972 by Ritchie and Thompson
  - Used to develop Unix operating system and Unix commands
  - Replacement for assembly language for hardware interface.
  - By late 1970's C had evolved to "K & R C"
- C Standards
  - 1st C standard created in 1989 by ANSI, ratified by ISO in 1990.

    It is called C89. Some call it C90.
  - 2nd C standard was ratified in 1999, called C99.
  - 3rd and current standard is C11

Just the Facts

- C was developed in 1972 in order to write the UNIX operating system
- C is more "low level" than other high level languages (good for MCU programming)
- C is supported by compilers for a wide variety of MCU architectures
- C can do *almost* anything assembly language can do
- C is usually easier and faster for writing code than assembly language

Busting the Myths (1.2)

The truth shall set you free…

- C is not as portable between architectures or compilers as everyone claims
  - ANSI language features ARE portable
  - Processor specific libraries are NOT portable
  - Processor specific code (peripherals, I/O, interrupts, special features) are NOT portable

- C is NOT as efficient as assembly
  - A *good* assembly programmer can *usually* do better than the compiler, no matter what the optimization level – C WILL use more memory

Busting the Myths (2.2)

The truth shall set you free…

- There is NO SUCH THING as self documenting code – despite what many C proponents will tell you
  - C makes it possible to write very confusing code – just search the net for obfuscated C code contests… (www.ioccc.org)

- C is not always friendly to new users – hence the need for comments!

Opaque C code:

```
void duff(register char *to,
   register char *from, register int count)
{
    register int n=(count+7)/8;
    switch(count%8){
    case 0: do{ *to++ = *from++;
    case 7:  *to++ = *from++;
    case 6: *to++ = *from++;
    case 5: *to++ = *from++;
    case 4: *to++ = *from++;
    case 3: *to++ = *from++;
    case 2: *to++ = *from++;
    case 1: *to++ = *from++;
            }while( --n >0);
    }
}
```

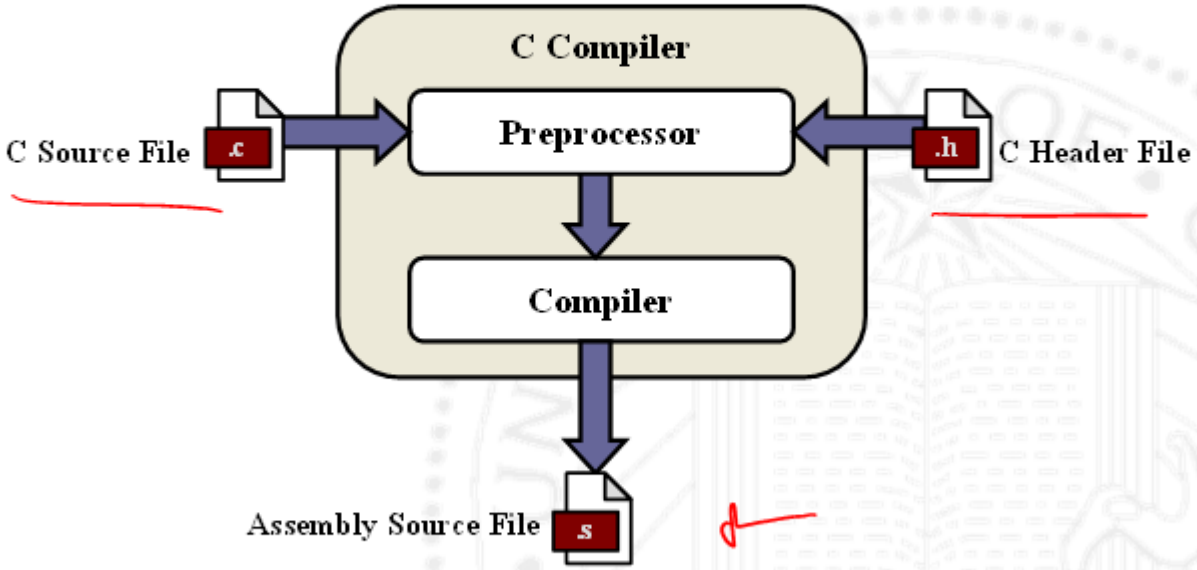
```
int count=0;
while(x)
{
    count++;
    x = x&(x-1);
}
return count;
```

```
#define offsetof(a,b) ((int)(&(((a*)(0))->b)))
```

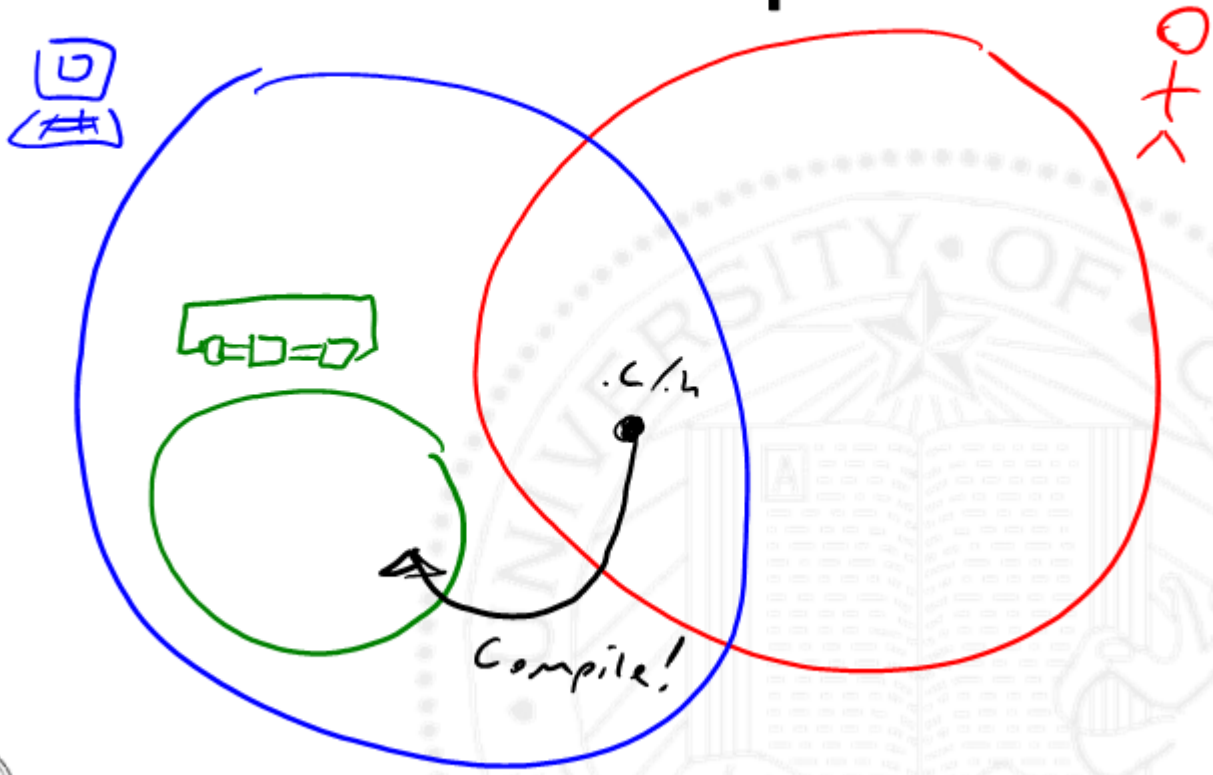
http://www.gowrikumar.com/c/index.php

How does C work?

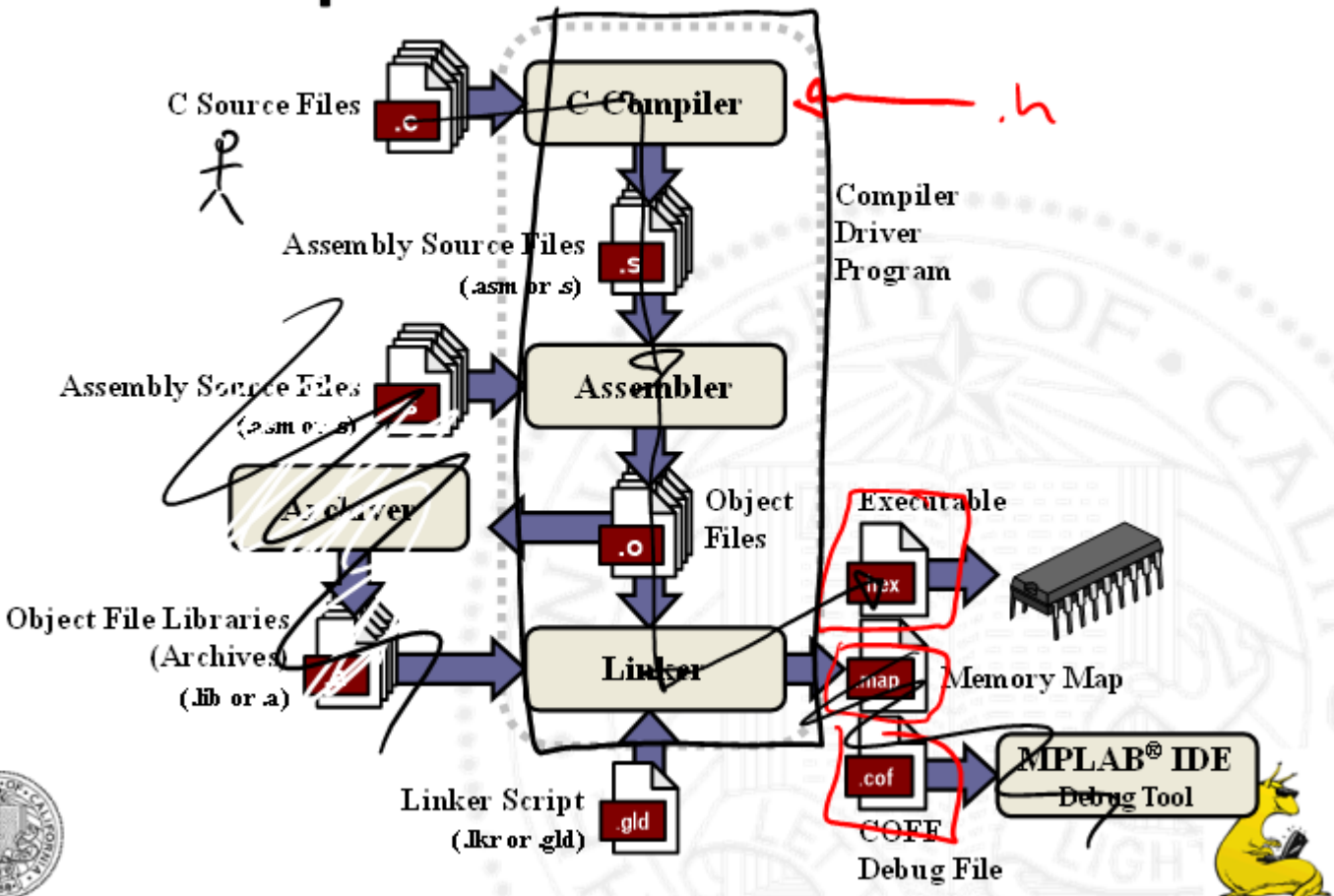
Development Tools Data Flow



What is a compiler?



Development Tools Data Flow



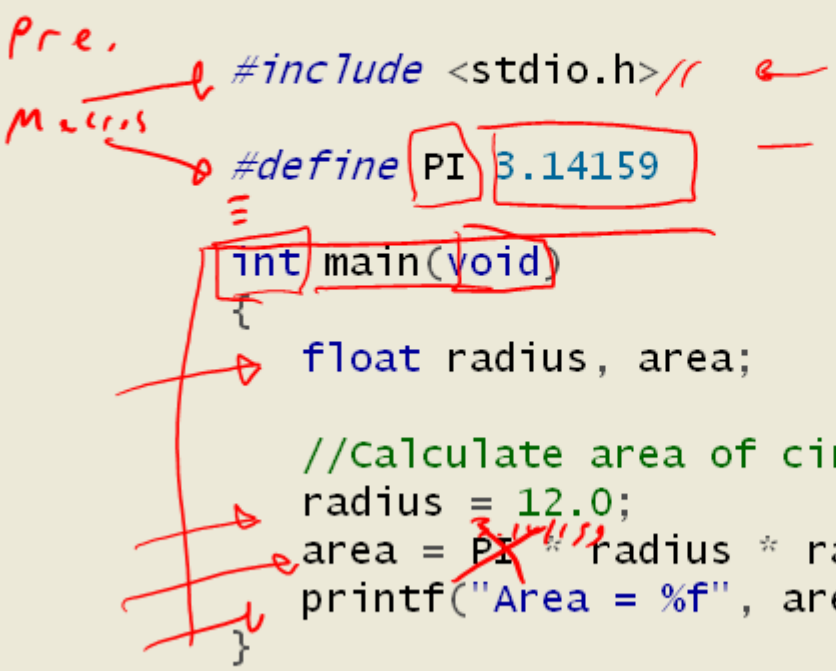
Compilation

- **Source Code Analysis**
  - "front end"
  - parses programs to identify its pieces
  - variables, expressions, statements, functions, etc.
  - depends on language (not on target machine)
- **Code Generation**
  - "back end"
  - generates machine code from analyzed source
  - may optimize machine code to make it run more efficiently
  - dependent on target architecture
- **Symbol Table**
  - map between symbolic names and items
  - like assembler, but more kinds of information

Fundamentals of C

A Simple C Program

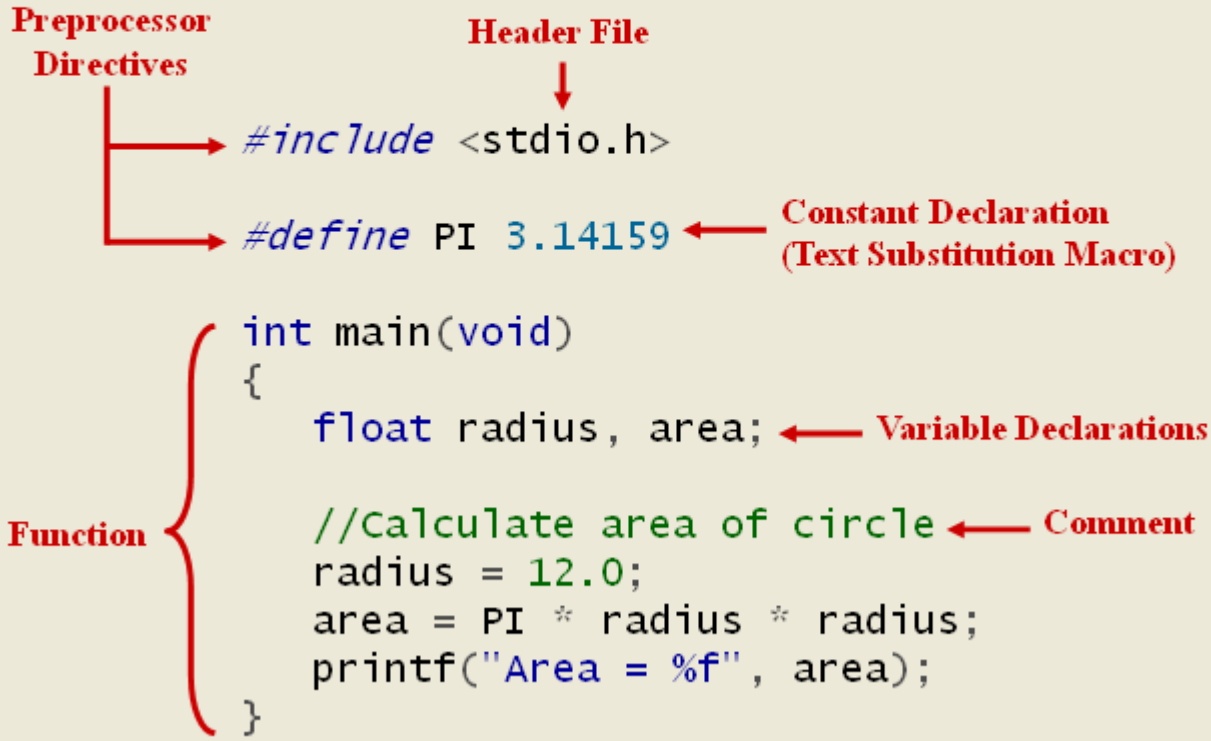
Example



Fundamentals of C

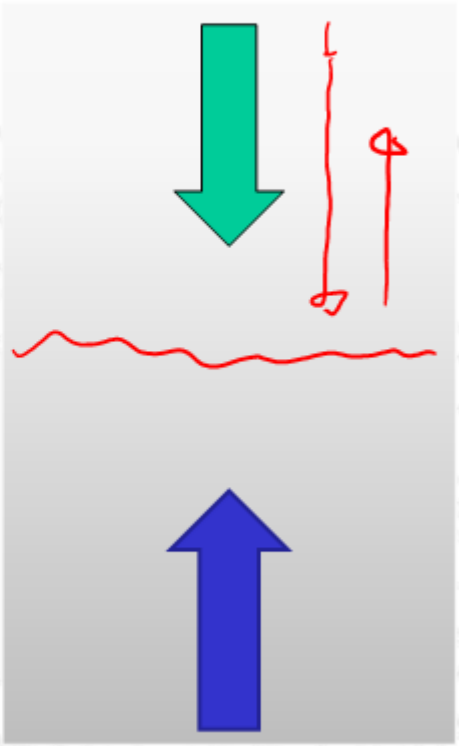
A Simple C Program

Example

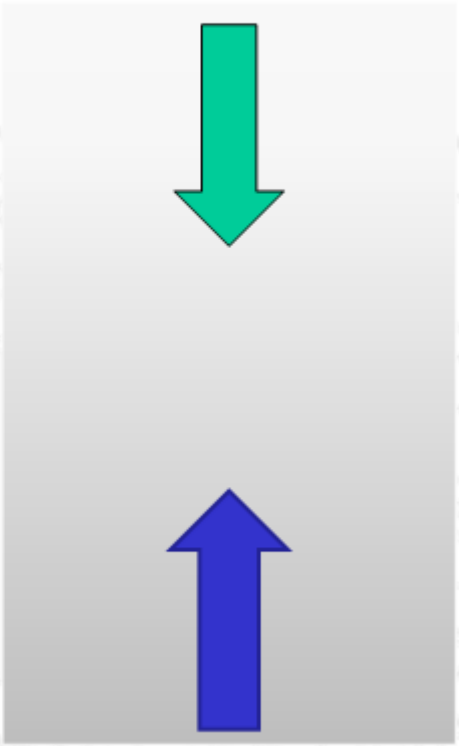


Stack/Heap

- Stack grows from bottom “up”
- Heap grows from top “down”



Stack/Heap



Basic Elements of C

C Runtime Environment (1.2)

- C Runtime is the "backend" of C:
  - Allocates space for stack
  - Initialize stack pointer
  - Allocates space for heap
  - Copies values from Flash/ROM to variables in RAM that were declared with initial values
  - Clear uninitialized RAM
  - Disable all interrupts
  - Call main() function (where your code starts)

C Runtime Environment (2.2)

- Runtime environment setup code is automatically linked into your application by XC32 compiler
- Code comes from:
  - XC32: crt0.s / crt0.o (crt = C RunTime)
- User modifiable if absolutely necessary

Key Concepts of CE13

What we will cover in 13/L (1/3)

- **"C" programming**
  - Using C in an Embedded Environment
  - Variables, Identifiers and Data Types
  - printf() and scanf()
  - Operators
  - Preprocessor Macros
  - Expressions and Statements
  - Making Decisions
  - Loops
  - Functions
  - Scope
  - Multi-File Projects

What we will cover in 13/L (2/3)

- **Data structures**
  - Arrays ✓
  - Data Pointers ✓
  - Function Pointers ✗
  - Structures ✓
  - Unions ✓
  - Bit Fields ✓
  - Typedef
- **General Techniques**
  - State Machines ✓
  - Recursion ✓
  - Interrupts ✓
  - Program decomposition ✓
  - Abstraction ✓
  - Static / Dynamic Memory allocation ✓

9

What we will cover in 13/L (3/3)

- **Embedded "C" on a microcontroller**
  - Specific issues with uControllers
  - Peripheral usage
  - Reading documentation
- **Testing and Debugging**
  - Commenting
  - Test harnesses
  - Incremental development
  - Issues with embedded debugging

Key things we'll enforce (1.2)

- **Well documented code**
  - Self-documenting code does not exist!
  - Good variable names and comments are req'd
- **Clean and clear style**
  - More important to adhere to established guidelines than use the "right" style
  - We'll use (modified) K&R style guide
- **Modularity and decomposition**
  - Code is segmented by functionality
  - Proper use of .h and .c files
  - Good use of functions for clean implementation

Key things we'll enforce (2.2)

- **State Machines / Event Driven Programming**
  - Best way to design reactive systems
  - Makes debugging much easier
- **Incremental build and test**
  - Every bit of code has a unit test
  - Unit test is designed along with code block
  - Use of pseudo-code and comments
  - End to end code checks

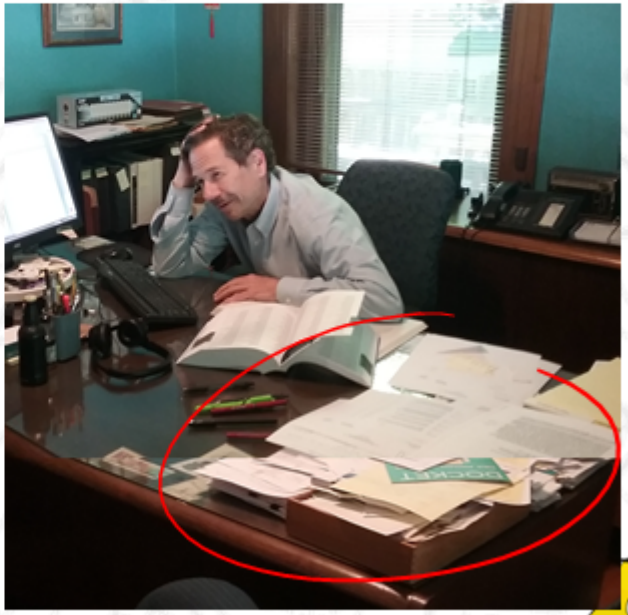
Ugly/Beautiful

Ugly Garden

- Beautiful Garden

Ugly/Beautiful

Ugly Desk



- Beautiful Desk

Ugly/Beautiful

- 

- Beautiful Fridge

Ugly Fridge

Ugly/Beautiful

```
/*****************************************************************************
 Function: RC_Init

 Parameters
     RCpins: an unsigned short with a 1 in each position to set the pin as an RC
             servo pin, should be a bitwise OR of the #define'd RC_PORTxxx pins.

 Returns
     char: SUCCESS or ERROR

 Description
    Initializes the RC_Servo subsystem, sets each pin as a digital output, and
    sets the uptime for each pin at 1.5msec, with a period of 20msec.

 Notes: Uses TIMER4 with a rollover.

 Author: Gabriel Hugh Elkaim, 2011.12.15 16:42
 *****************************************************************************/
char RC_Init(unsigned short int RCpins) {
    char i, curPin;
    unsigned short int CurrentTime;
    dbprintf("\nInitializing the RC Servo Module.");
    // Check if inputs are in range, and if already initialized
    if ((RCpins == 0x000) || (RCpins > 0x2FF) || (RCstate != off)) {
        return ERROR;
    }
    RCstate = init;
    // Go through input and set each RC pin direction and force to low
    for (i = 0; i < RCPINCOUNT; i++) {
        curPin = (1 << i);
        if (RCpins & curPin) {
            RCpinMap[numRCPins] = i;
            numRCPins++;
            RC_upTime[i] = SERVOCENTER;
            *RC_TRISCLR[i] = rcBitsMap[i]; // Sets pin direction to output
            *RC_LATCLR[i] = rcBitsMap[i]; // Forces pin to low state
            dbprintf("\nEnabling pin: 0x%X", curPin);
        }
```

```
typedef char C;typedef long I;
typedef struct a{I t,r,d[3],p[2];}*A;
#define P printf
#define R return
#define V1(f) A f(w)A w;
#define V2(f) A f(a,w)A a,w;
#define DO(n,x) {I i=0,_n=(n);for(;i<_n;++i){x;}}
I *ma(n){R(I*)malloc(n*4);}mv(d,s,n)I *d,*s;{DO(n,d[i]=s[i]);}
tr(r,d)I *d;{I t=1;DO(r,t=t*d[i]);R t;}
A ga(t,r,d)I *d;{A t=(A)ma(5+tr(r,d));t->t=t,t->r=r,mv(t->d,d,r);
 R t;}
V1(iota){I n=*w->p;A t=ga(0,1,&n);DO(n,t->p[i]=i);R t;}
V2(plus){I r=w->r,*d=w->d,n=tr(r,d);A t=ga(0,r,d);
 DO(n,t->p[i]=a->p[i]+w->p[i]);R t;}
V2(from){I r=w->r-1,*d=w->d+1,n=tr(r,d);
 A t=ga(w->t,r,d);mv(t->p,w->p+(n**a->p),n);R t;}
V1(box){A t=ga(1,0,0);*t->p=(I)w;R t;}
V2(cat){I an=tr(a->r,a->d),wn=tr(w->r,w->d),n=an+wn;
 A t=ga(w->t,1,&n);mv(t->p,a->p,an);mv(t->p+an,w->p,wn);R t;}
V2(find){}
V2(rsh){I r=a->r?*a->d:1,n=tr(r,a->p),wn=tr(w->r,w->d);
 A t=ga(w->t,r,a->p);mv(t->p,w->p,wn=n>wn?wn:n);
 if(n-=wn)mv(t->p+wn,t->p,n);R t;}
V1(sha){A t=ga(0,1,&w->r);mv(t->p,w->d,w->r);R t;}
V1(id){R w;}V1(size){A t=ga(0,0,0);*t->p=w->r?*w->d:1;R t;}
pi(i){P("%d ",i);}nl(){P("\n");}
pr(w)A w;{I r=w->r,*d=w->d,n=tr(r,d);DO(r,pi(d[i]));nl();
 if(w->t)DO(n,P("< ");pr(w->p[i]))else DO(n,pi(w->p[i]));nl();}

C vt[]="+{~<#,";
A(*vd[])()={0,plus,from,find,0,rsh,cat},
 (*vm[])()={0,id,size,iota,box,sha,0};
```

Concept of Ugly/Beautiful

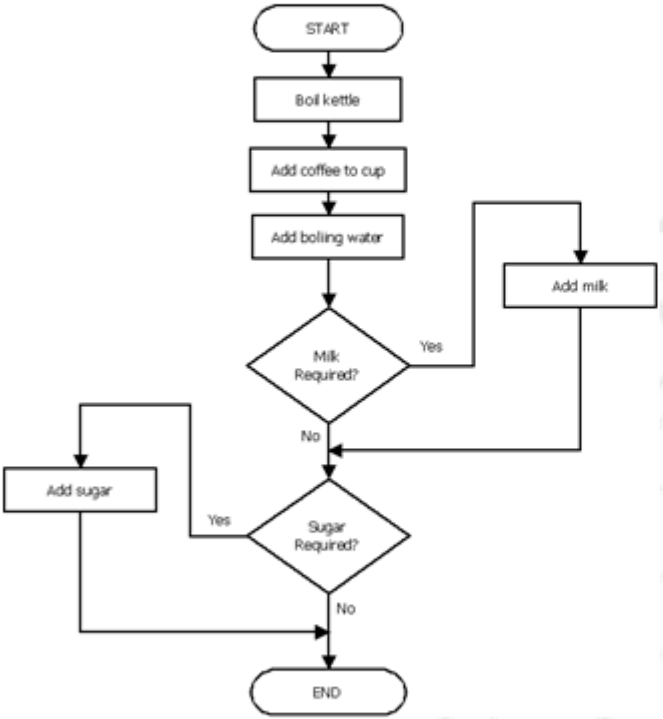
- Code can be ugly or beautiful
- Beautiful code IS better code!

- While you might not recognize it, yet, you will by the end of this quarter.

Software Architecture

- The “shape” of your code
- Often orthogonal to Project Management
- Can be the success or failure of a project
- Write it BEFORE you start writing code

Flow Chart



- Flowcharts are visual representations of an algorithm
- These allow you to think about the structure of your code more naturally

Pseudo-code

Particle_filter$(\mathcal{X}_{t-1}, u_t, z_t)$:

1: $\bar{\mathcal{X}}_t = \mathcal{X}_t = \emptyset$
2: *for* $m = 1$ *to* M *do*
3: *sample* $x_t^{[m]} \sim \pi(x_t)$
4: $w_t^{[m]} = \frac{p(x_t^{[m]})}{\pi(x_t^{[m]})}$
5: $\bar{\mathcal{X}}_t = \bar{\mathcal{X}}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$
6: *endfor*
7: *for* $m = 1$ *to* M *do*
8: // *draw* i *with probability* $\propto w_t^{[i]}$
9: *add* $x_t^{[i]}$ *to* $\mathcal{X}_t$
10: *endfor*
11: *return* $\mathcal{X}_t$

- Pseudo-code is plain English that explains in coarse steps what the code should do
  - Doesn't use rigorous syntax
  - Hides details of programming language
  - A great way to start a new piece of code: Paste in pseudo-code as comments, then fill it in

Style Examples

- Adherence to a specific style
  - Variables in camelCase with leading lower case
  - Functions in CamelCase with leading Upper case
  - #define (literal constants) are in UPPERCASE
    - Exception: macros can look like functions if they act like one
  - Variable and Function names are descriptive
    - Eg: backingUpState = TRUE;
  - Correct use of white space and indentation
  - Correct placement of braces {}

Roadmap

- Hello and Welcome
- CMPE13 Overview ✓
- What is C? ✓
- How does C work? ✓
- Basic elements of a C program ✓
- Key Concepts of CE13 ✓
- Lab01 / git demo (?)

Questions?