ECE 250 / CS 250
Computer Architecture

“C Programming”

Benjamin Lee

Some slides based on those from Alvin Lebeck, Daniel Sorin, Andrew Hilton, Amir Roth, Gershon Kedem

Outline

- Previously:
  - Computer is a machine that does what we tell it to do
- Next:
  - How do we tell computers what to do?
    - » First a quick intro/review of C programming
    - » Goal: to learn C, not teach you to be an expert in C
  - How do we represent data?
  - What is memory?

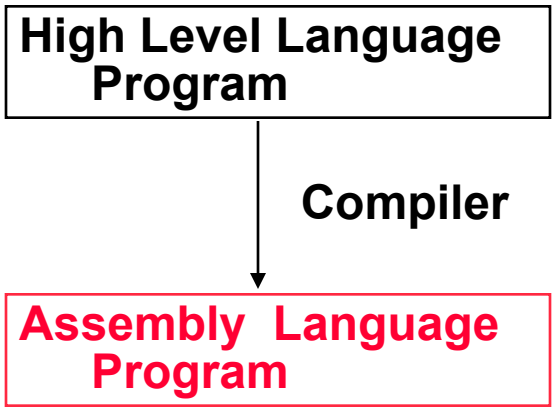
We Use High Level Languages

High Level Language Program

```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

- There are many high level languages (HLLs)
  - Java, C, C++, C#, Fortran, Basic, Pascal, Lisp, Ada, Matlab, etc.
- HLLs tend to be English-like languages that are "easy" for programmers to understand
- In this class, we'll focus on C as our running example for HLL code. Why?
  - C has pointers
  - C has explicit memory allocation/deallocation
  - Java hides these issues (don't get me started on Matlab)

HLL → Assembly Language

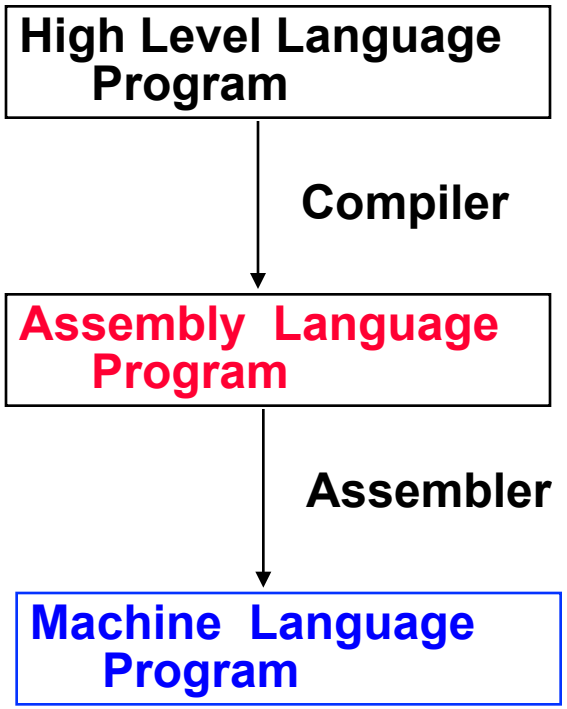



```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

```
lw      $15,   0($2)
lw      $16,   4($2)
sw      $16,   0($2)
sw      $15,   4($2)
```

- Every computer architecture has its own assembly language
- Assembly languages tend to be pretty low-level, yet some actual humans still write code in assembly
- But most code is written in HLLs and compiled
  - Compiler is a program that automatically converts HLL to assembly

Assembly Language → Machine Language



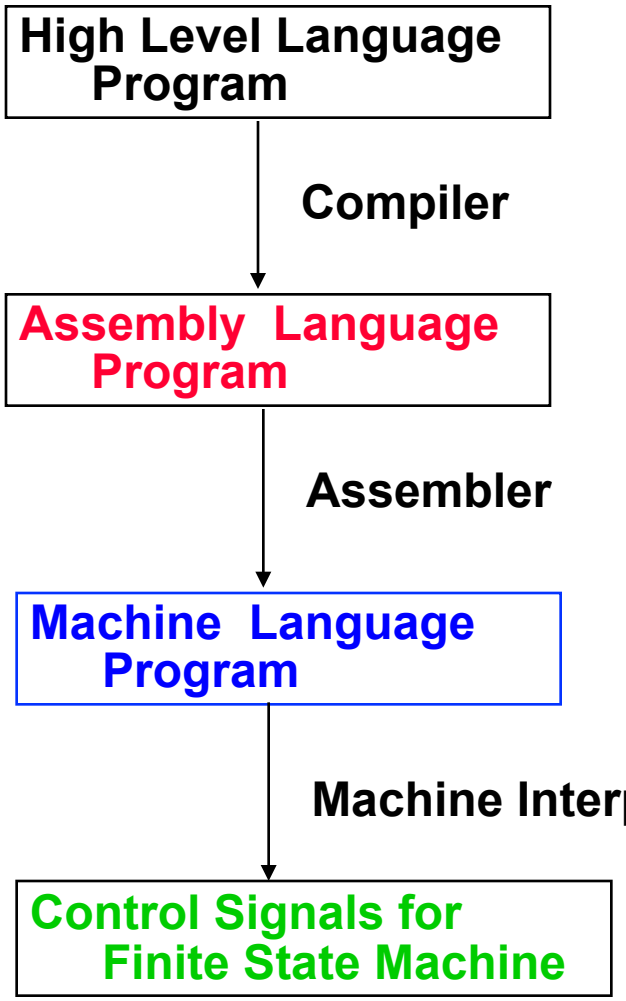

```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

```
lw      $15,   0($2)
lw      $16,   4($2)
sw      $16,   0($2)
sw      $15,   4($2)
```

```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
```

- Assembler program automatically converts assembly code into the binary machine language (zeros and ones) that the computer actually executes

Machine Language → Inputs to Digital System




```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

```
lw      $15,   0($2)
lw      $16,   4($2)
sw      $16,   0($2)
sw      $15,   4($2)
```

```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
```

Transistors (switches) turning on and off

What you know today

JAVA

```
...
System.out.println("Please Enter In Your First Name: ");
String firstName = bufRead.readLine();
System.out.println("Please Enter In The Year You Were Born: ");
String bornYear = bufRead.readLine();
System.out.println("Please Enter In The Current Year: ");
String thisYear = bufRead.readLine();
int bYear = Integer.parseInt(bornYear);
int tYear = Integer.parseInt(thisYear);
int age = tYear - bYear ;
System.out.println("Hello " + firstName + ". You are " + age + " years
old");
```

How does a Java program execute?

- Compile Java Source to Java Byte codes
- Java Virtual Machine (JVM) interprets/translates Byte codes
- JVM is a program executing on the hardware

- Java has lots of things that make it easier to program without making mistakes

- JVM handles memory for you
  - What do you do when you remove an entry from a hash table, binary tree, etc.?

The C Programming Language

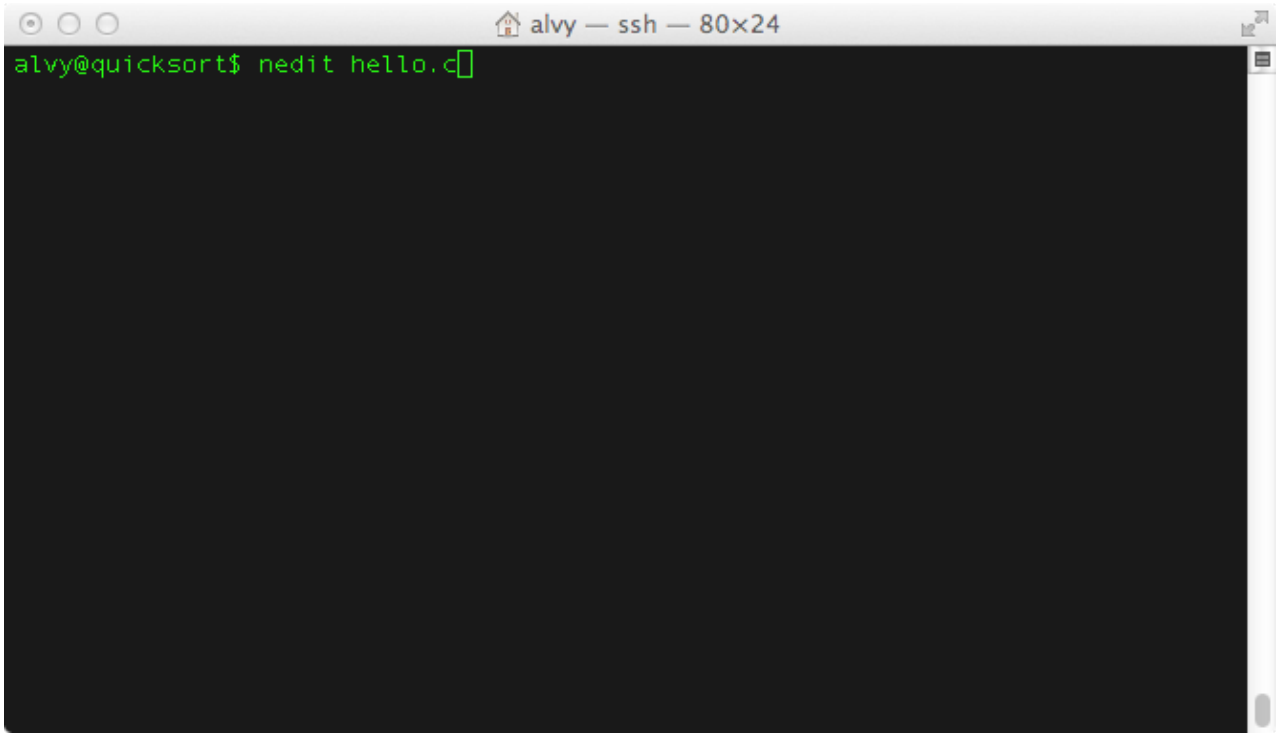
- No virtual machine
  - No dynamic type checking, array bounds, garbage collection, etc.
  - Compile source file directly to machine
- Closer to hardware
  - Easier to make mistakes
  - Can often result in faster code
- Generally used for ‘systems programming’
  - operating systems, embedded systems, database implementation
  - There is object oriented C++ (C is a strict subset of C++)

The C Programming Language (Continued)

- No objects
- Procedural, not object oriented
  - No objects with methods
- Structures, unions like objects
  - Member variables (no methods)
- Pointers – memory, arrays
- External standard library – I/O, other facilities
- Macro preprocessor (#<directive>)
- Resources
  - Kernighan & Richie book *The C Programming Language*
  - MIT open course *Practical Programming in C* in 'docs' of website
  - Drew Hilton Video Snippets

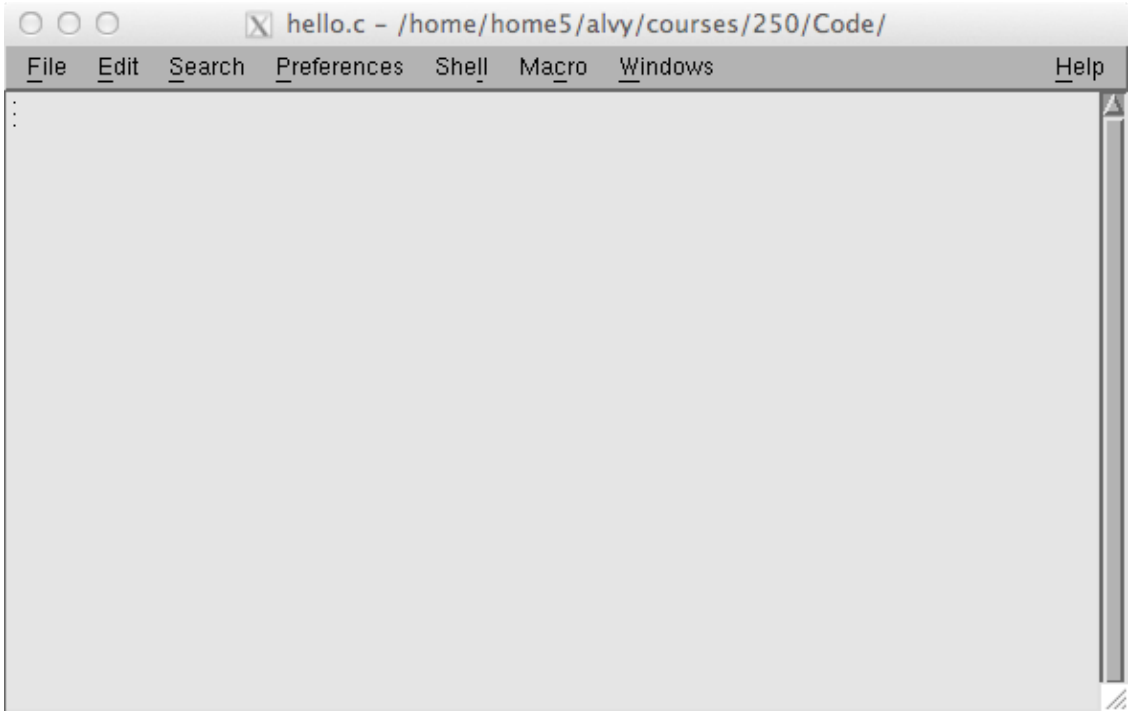
Creating a C source file

- We are not using a development environment (IDE)
- You will create programs starting with an empty file!
- Use .c file extension (e.g., hello.c)
- On a linux machine you can use nedit



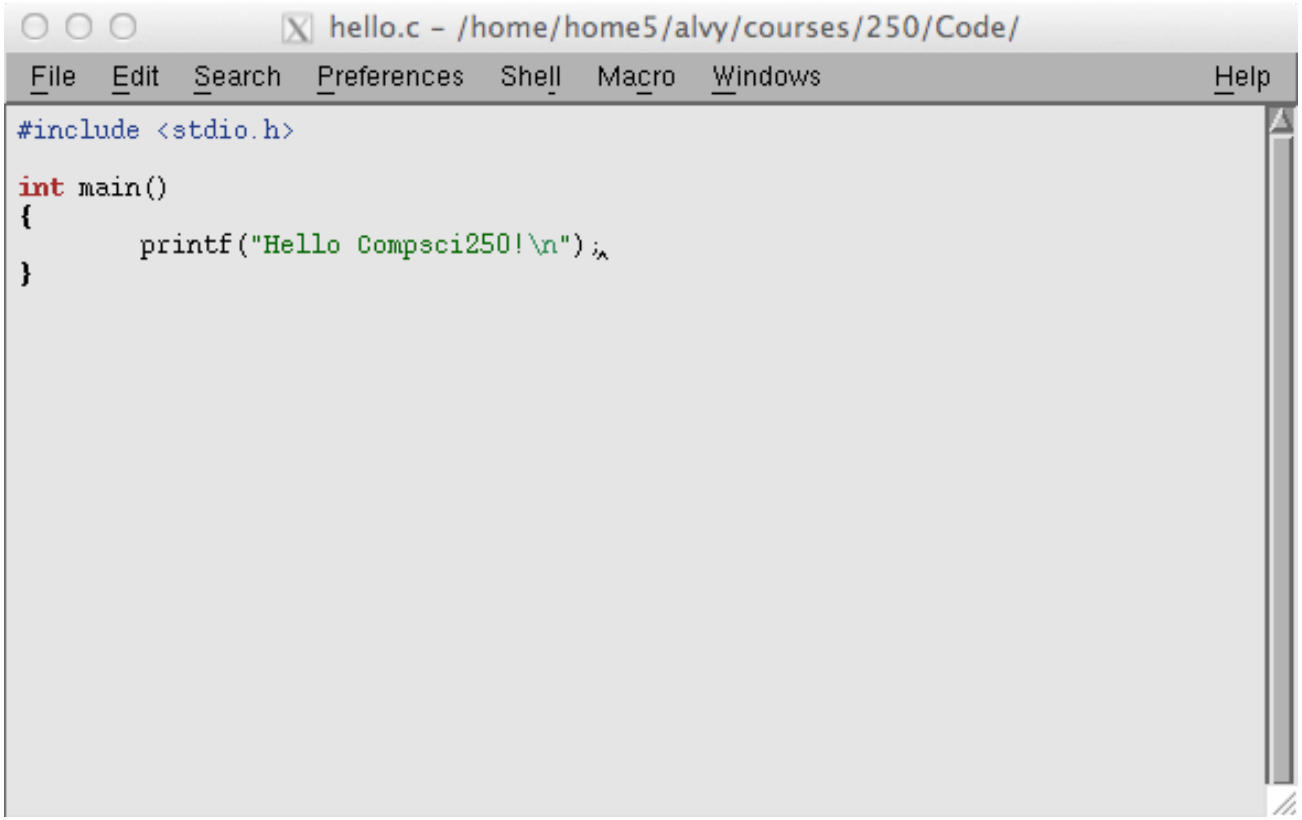
The nedit window

- nedit is a simple point & click editor
  - with ctrl-c, ctrl-x, ctrl-v, etc. short cuts
- Feel free to use any text editor (gvim, emacs, etc.)



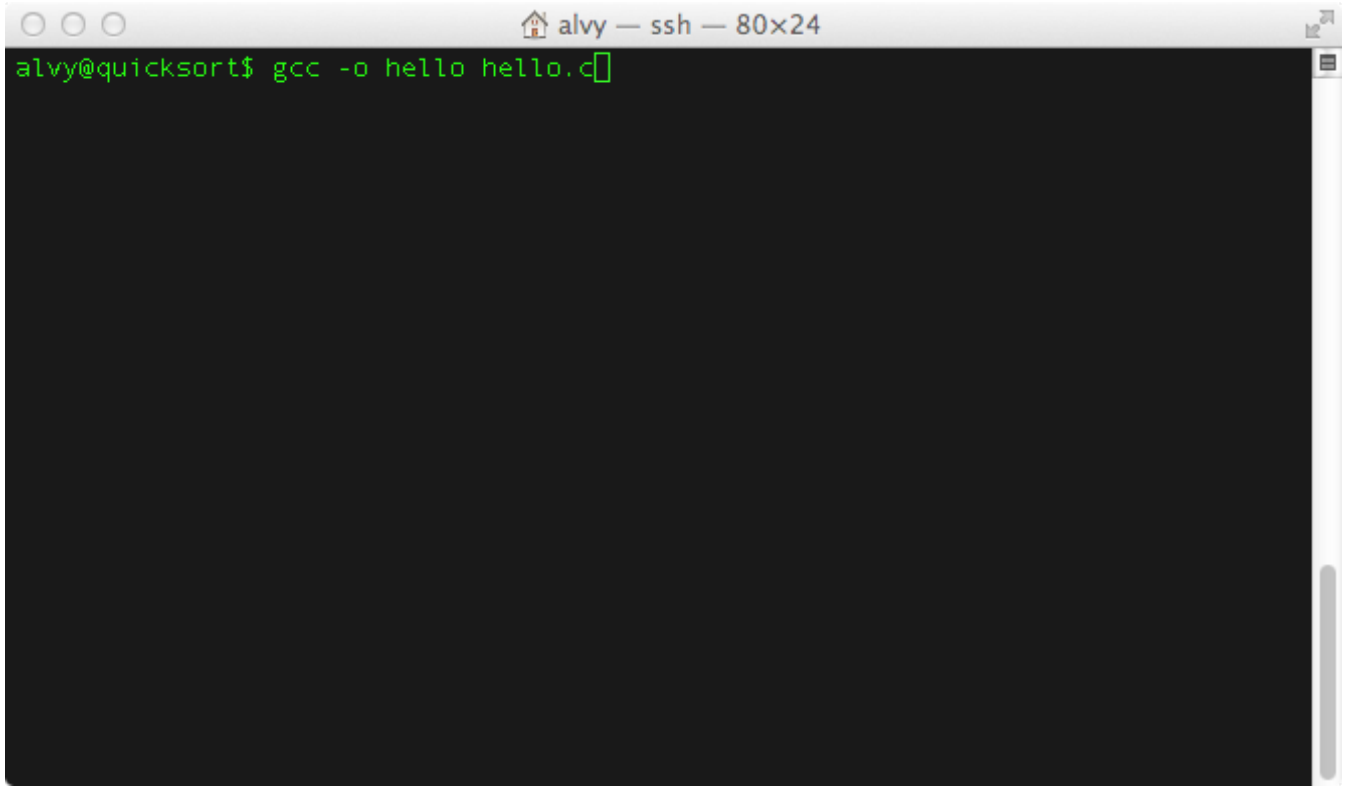
Hello World

- Canonical beginner program
  - Prints out “Hello …”
- nedit provides syntax highlighting



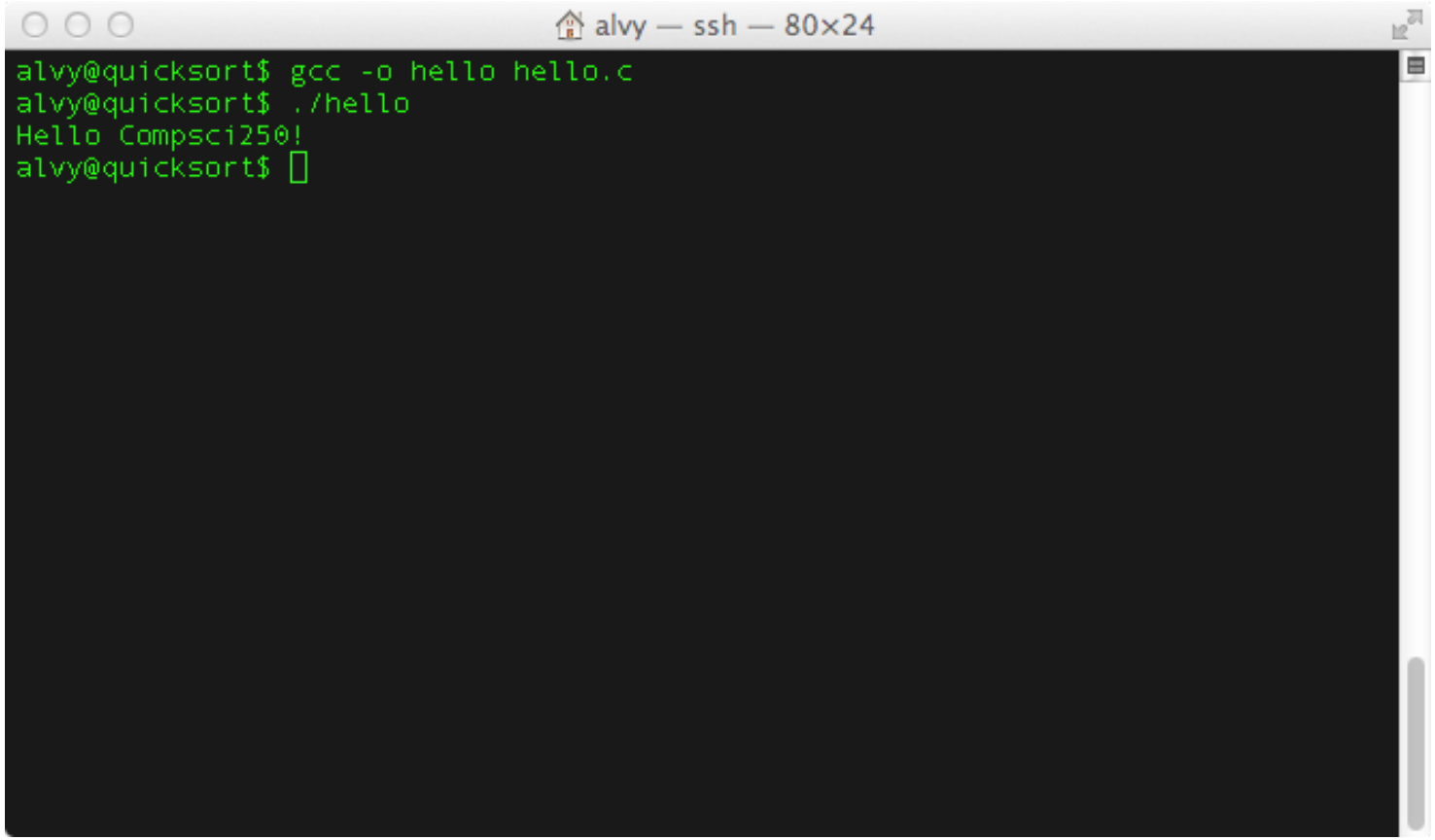
Compiling the Program

- Use the gcc program to create an executable file
- gcc –o <outputname> <source file name>
- gcc –o hello hello.c (must be in same directory as hello.c)
- If no –o option, then default output name is a.out (e.g., gcc hello.c)



Running the Program

- Type the program name on the command line
  - ./ before “hello” means look in current directory for hello program



Debugging

- Print debugging…
  - Just output information at different points in the program
  - Not the most efficient, but often works.

- gdb <executable filename>
- Good for stopping at set points in program and inspecting variable values.
  - If you get good at using a debugger it is easier/better than printf debugging…
- Recitation for more on debugging

Variables, operators, expressions

- C variables are similar to Java
  - Data types: int, float, double, char, void
  - Signed and unsigned int
  - char, short, int, long, long long can all be integer types
    - » These specify how many bits to represent an integer
- Constants
  - Use #define C preprocessor
  - E.g.,: #define MAX_SCORE 100
- Operators:
  - Mathematical +, -, *, /, %,
  - Logical !, &&, ||, ==, !=, <, >, <=, >=
  - Bitwise &, |, ~, ^ , <<, >> (we'll get to what these do later)
- Expressions: var1 = var2 + var3;

Variables Continued

- Must be declared before use
- Remember to initialize

- Can initialize at declaration
  - int n = 23;
  - int a, b c, d = 0;
  - float foo = 3.141;

- What value does an uninitialized variable have?

Type Conversion

- Use type casting to convert between types
  - variable1 = (new type) variable2;
  - Note order of operations – cast often takes precedence

(double) a (int) (x+y) (int) x+y

```
main() {
  float x;
  int i;
  x = 3.6;
  i = (int) x;
  printf("x=%f, i=%d", x, i)
}
```

result: x=3.600000, i=3

Control Flow

- Conditionals

  If (a < b) { … } else {…}

  switch (a) {

  ```
      case 0: s0; break
      case 1: s1;
      case 2: s2; break
      default: break;
  ```

  }

- Loops

  for (i = 0; i < max; i++) { ... }

  while (i < max) {…}

Functions

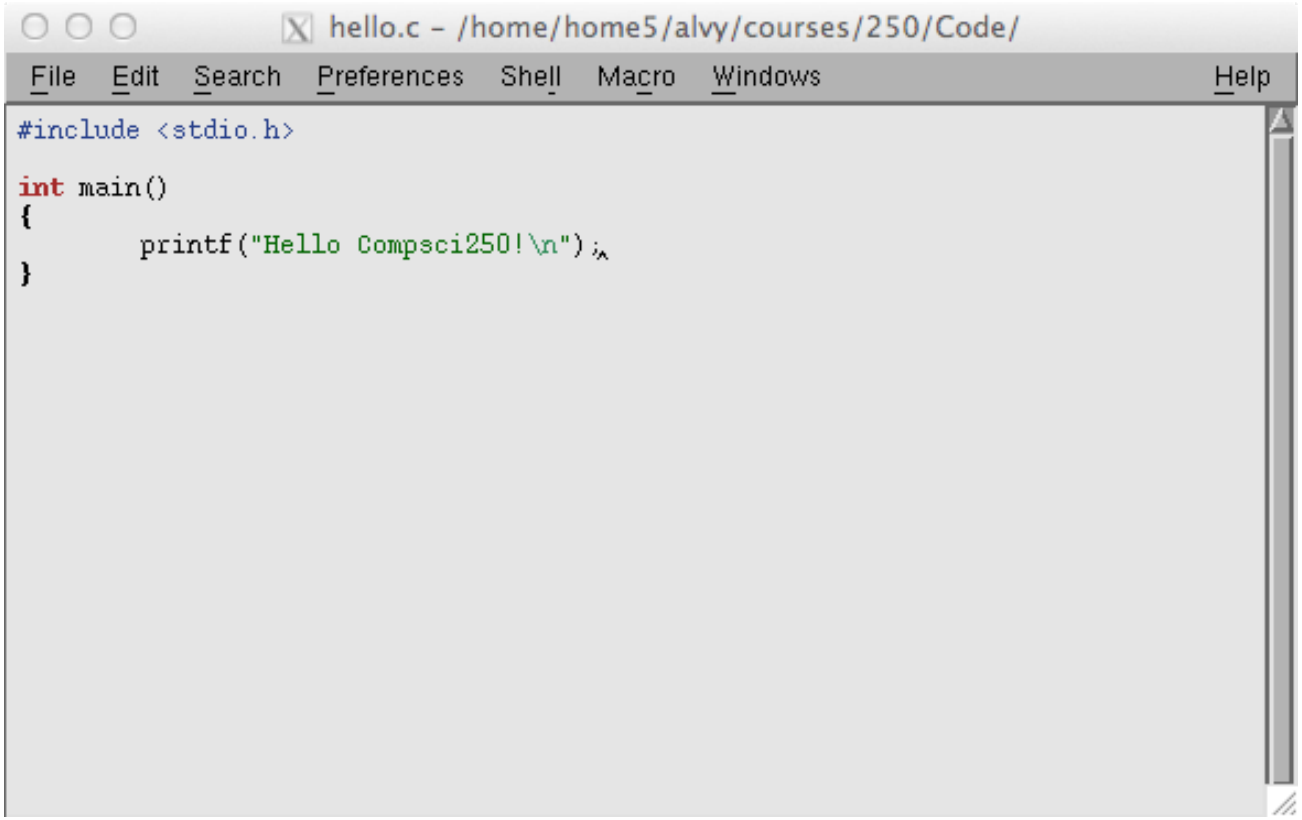
- Encapsulate computation
  - Reuse or clarity of code
  - Cannot define functions within functions
- Must be declared before use!

```
int div2(int x,int y); /* declaration here */
main() {
        int a;
        a = div2(10,2);
}
int div2(int x, int y) {  /* implementation here */
        return (x/y);
}
```

- Or put functions at top of file (doesn't always work)

Back to our first program

- #include <stdio.h> defines input/output functions in C standard library
- printf(…) writes output to terminal



Input Output

- Read/Write to/from the terminal
  - Standard input, standard output (defaults are terminal)
- Character I/O
  - putchar(), getchar()
- Formatted I/O
  - printf(), scanf()

Character I/O

```
#include <stdio.h>  /* include the standard IO library function defs */
int main()
{
        char c;
        while ((c = getchar()) != EOF ) {   /* read characters until end of file */
              if (c == 'e')
                    c = '-';
              putchar(c);
        }
        return 0;
}
```

- EOF is End Of File (type ^d)
- What does the following command line do?
  - ./a.out < in.txt > out.txt

Formatted I/O

```
#include <stdio.h>
int main()
{
        int a = 23;
        float f =0.31234;
        char str1[] = “satisfied?”;
        /* some code here… */
        printf(“The values are %d, %f , %s\n”, a, f, str1);
        scanf(“%d %f”, &a, &f);  /* will come back to the & later */
        scanf(“%s”, str1);
        printf(“The values are %d, %f , %s\n”,a,f,str1);
}
```

- printf(“format string”, v1,v2,…);
  - \n is newline character
- scanf(“format string”,…);
  - Returns number of matching items or EOF if at end-of-file

Reading Input in a Loop

```
#include <stdio.h>
int main()
{
        int an_int = 0;
        while(scanf("%d",&an_int) != EOF) {
                printf("The value is %d\n",an_int);
        }
}
```

- This reads integers from the terminal until the user types ^d (ctrl-d)
  - Can use a.out < file.in
- WARNING THIS IS NOT CLEAN CODE!!!
  - If the user makes a typo and enters a non-integer it will loop indefinitely!!!
- How to stop a program that is in an infinite loop on Linux?
- Type ^c (ctrl-c) It kills the currently executing program.
- Type “man scanf” on a linux machine and you can read a lot about scanf
- See MIT open courseware notes or web on input/output

Global Variables

- Global variables are accessible from any function

```
#include <stdio.h>
int a = 0;
float f = 0;
void seta() { a = 78; }
int main()
{
        a = 23;
        f =0.31234;
        seta();
        printf("The values are %d, %f \n",a,f);
}
```

- What is the output?
  - what if in main we had "int a = 23;" ?

Header Files, Separate Compilation, Libraries

- C preprocessor
  - #include filename just inserts that file (like #include <stdio.h>)
  - #define MYFOO 8, replaces MYFOO with 8 in entire program
    - » Good for constants
    - » #define MAX_STUDENTS 100
- Separate Compilation
  - Many source files (e.g., main.c, students.c, instructors.c, deans.c)
  - gcc –o prog main.c students.c instructors.c deans.c
  - Produces one executable program from multiple source files
  - A bit more later, lots of good uses, but beyond this class
- Libraries: Collection of common functions (some provided, you can build your own)
    - » libc has io, strings, etc.
    - » libm has math functions (pow, exp, etc.)
    - » gcc –o prog file.c –lm (says use math library)
    - » You can read more about this elsewhere

Arrays

- Mostly the same as other languages
  - char buf[256];
  - int ar[256][512]; /* two dimensional array */
  - float scores[4196];
  - double speed[100];

```
for (i = 0; i< 256; i++)
        buf[i] = 'A'+i;
```

- Strings
  - char str1[256] = "hi";
  - str1[0] = 'h', str1[1] = 'i',str1[2] = 0;
  - 0 is value of NULL character '\0', identifies end of string
- What is C code to compute string length?

Compute String Length

```
int len=0;
while (str1[len] != 0)
        len++;
```

- Length does not include the NULL character
- C has built-in string operations
  - #include <string.h>
  - strlen(str1);

Structures

- Loosely like objects
  - Have member variables
  - Do not have methods!
- Structure definition with struct keyword

```
struct student_record {
        int id;
        float grade;
} rec1, rec2;
```

- Declare a variable of the structure type with struct keyword

```
struct student_record onerec;
```

- Access the structure member fields with '.' structvar.member

```
onerec.id = 12;
onerec.grade = 79.3;
```

Array of Structures

```
#include <stdio.h>
struct student_record {
        int id;
        float grade;
};
struct student_record myroster[100];  /* declare array of structs */
int main()
{
        myroster[23].id = 99;
        myroster[23].grade = 88.5;
        printf("ID %d, grade %f\n",myroster[23].id,myroster[23].grade);
}
```

Reference vs. Pointer

Java

- "The value of a reference type variable, in contrast to that of a primitive type, is a reference to (an address of) the value or set of values represented by the variable"

  http://java.sun.com/docs/books/tutorial/java/nutsandbolts/datatypes.html

- Cannot manipulate value of reference

C

- Pointer is variable that contains location of another variable
- Pointer is memory location that contains address of another memory location
- Can manipulate value of pointer

Pointers

- Declaration of pointer variables
  - int * x_ptr; char * c_ptr; void * ptr;
- How do we get the location (address) of a variable?

Use one of the following:

1. Use the & 'address of' operator
   - x_ptr = &intvar;
2. From another pointer (yes we can do arithmetic on them)
   - x_ptr = y_ptr + 18;
3. Return from memory allocator
   - x_ptr = (int *) malloc(sizeof(int));

- More about addresses and pointers in a couple lectures…
  - char str1[256] is similar to str2 = (char *) malloc(256);

Pointers

- De-reference using *ptr to get what is pointed at

statement	x	x_ptr
int x;	??	??
int *x_ptr;	??	??
x = 2	2	??
x_ptr = &x;	2	&x
*x_ptr = 68;		
x_ptr = 200;		

Pointers

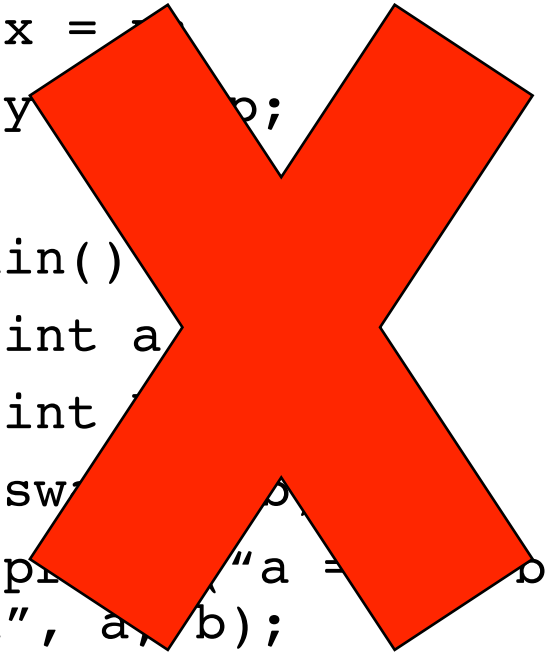
- De-reference using *ptr to get what is pointed at

statement	x	x_ptr
int x;	??	??
int *x_ptr;	??	??
x = 2	2	??
x_ptr = &x;	2	&x
*x_ptr = 68;	68	&x
x_ptr = 200;	68	200
*x_ptr = 42	68	200

- Be careful with assignment to a pointer variable
  - You can make it point anywhere…can be very bad
  - You will this semester likely experience a “segmentation fault”
  - What is 200?

Pass by Value vs. Pass by Reference

```
void swap (int x, int y){
  int temp = x;
  x =
  y
}
main()
  int a
  int
  sw
  pr
\n", a, b);
}
```



```
void swap (int *x, int
*y){
  int temp = *x;
  *x = *y;
  *y = temp;
}
main() {
  int a = 3;
  int b = 4;
  swap(&a, &b);
  printf("a = %d, b= %d
\n", a, b);
}
```

C Memory Allocation

- How do you allocate an object in Java?
- What do you do when you are finished with an object?
- Garbage collection
  - Counts references to objects, when == 0 can reuse
- C does not have garbage collection
  - Must explicitly manage memory
- void * malloc(nbytes)
  - Obtain storage for your data (like new in Java)
  - Often use *sizeof(type)* built-in returns bytes needed for type
  - Cast return value into appropriate type (int) malloc(sizeof(int));
- free(ptr)
  - Return the storage when you are finished (no Java equivalent)
  - ptr must be a value previously returned from malloc

Linked List

```
#include <stdio.h>
#include <stdlib.h>
struct list_ent {
        int id;
        struct list_ent *next;
};
main()
{
  struct list_ent *head, *ptr;
  head = (struct list_ent *)
        malloc(sizeof(struct list_ent));
  head->id = 66;
  head->next = NULL;

  ptr = (struct list_ent *)
        malloc(sizeof(struct list_ent));
  ptr->id = 23;
  ptr->next = NULL;
  head->next = ptr;

  printf("head id: %d, next id: %d\n",
        head->id, head->next->id);

  ptr = head;
  head = ptr->next;

  printf("head id: %d, next id: %d\n",
        head->id, ptr->id);
  free(head);
  free(ptr);
}
```

Command Line Arguments

- Parameters to main (int argc, char *argv[])
  - argc = number of arguments (0 to argc-1)
  - argv is array of strings
  - argv[0] = program name

```
main(int argc, char *argv[]) {
  int i;
  printf("%d arguments\n", argc);
  for (i=0; i< argc; i++)
  printf("argument %d: %s\n", i, argv[i]);
}
```

Summary

- C Language is lower level than Java
- Many things are similar
  - Data types
  - Control flow
- Some important differences
  - No objects!
  - Explicit memory allocation/deallocation
- Create and compile a program
- Up Next:
  - So what are those chars, ints, floats?
  - And what exactly is an address?

Resources

- See Course Web page Docs/Resources
  - (Note: Not the Sakai web page…)
- MIT Open Course
- Video snippets by Prof Drew Hilton in ECE
  - Doesn't work with Firefox (use Safari or Chrome)

Outline

- Previously:
  - Computer is machine that does what we tell it to do
- Next:
  - How do we tell computers what to do?
    - » First a quick intro to C programming
  - How do we represent data?
  - What is memory, and what are these so-called adresses?