ECE 250 / CPS 250
Computer Architecture

C Programming

Benjamin Lee

Slides based on those from

Andrew Hilton (Duke), Alvy Lebeck (Duke)
Benjamin Lee (Duke), and Amir Roth (Penn)

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 to 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

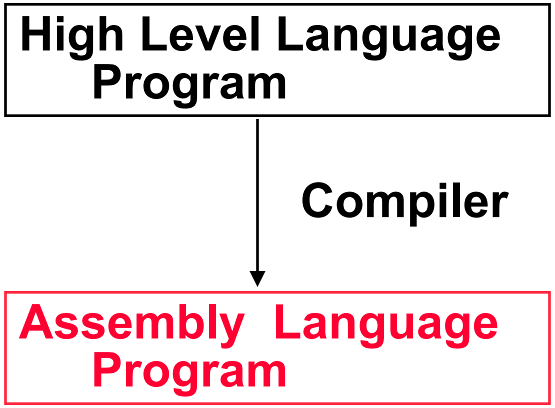
High Level Language Program

```
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 (will explain much more later)
  - C has explicit memory allocation/deallocation
  - Java hides these issues (don’t get me started on Matlab)

© Daniel J. Sorin
From Hilton, Lebeck, Lee, Roth

HLL → Assembly Language



High Level Language Program

Compiler

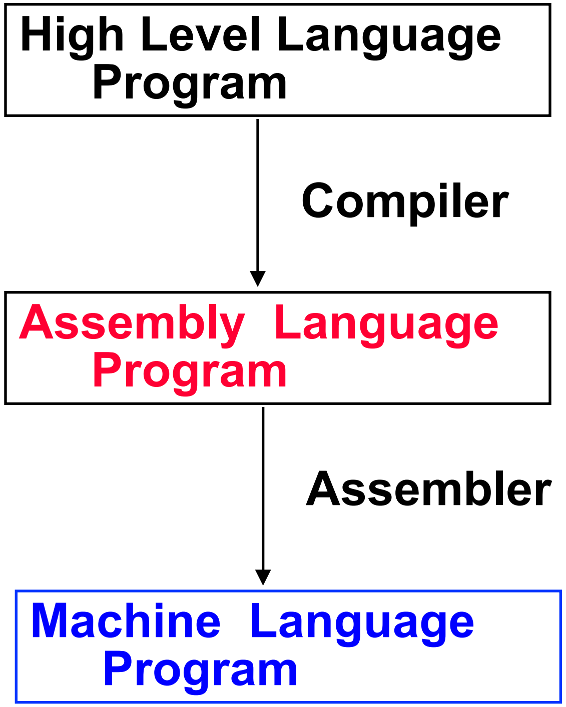
Assembly Language Program

```
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 an 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



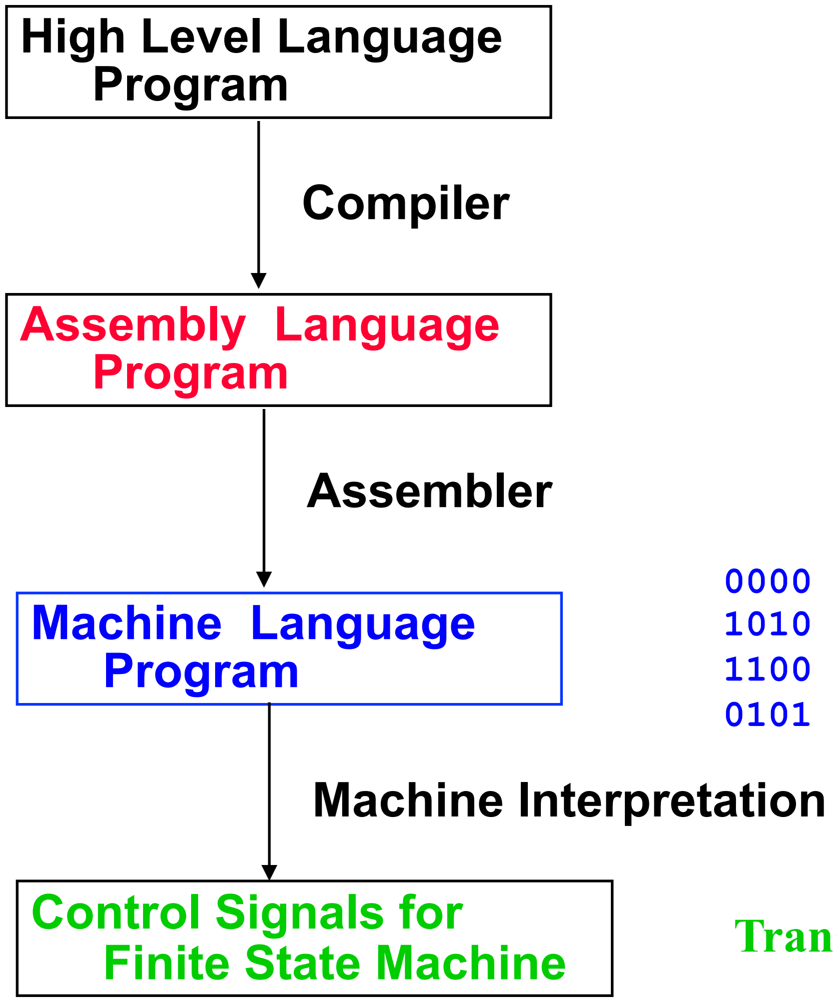
```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

```
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



High Level Language Program

Compiler

Assembly Language Program

Assembler

Machine Language Program

Machine Interpretation

Control Signals for Finite State Machine

```
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 features that make it easier to program without making mistakes → training wheels are nice

- 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 → training wheels slow you down

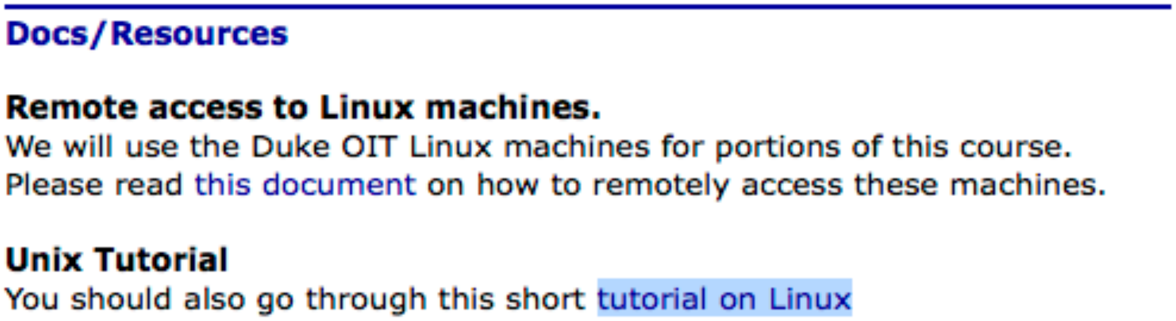
- Generally used for ‘systems programming’
  - Operating systems, embedded systems, database implementation
  - C++ is object-oriented version of C (C is a strict subset of C++)

Learning How to Program in C

- You need to learn some C
- I'll present some slides next, but nobody has ever learned programming by looking at slides or a book
  - You learn programming by programming!
- Goals of these slides:
  - Give you the big picture of how C differs from Java
  - Give you some important pointers to get you started
- Very useful resources
  - Kernighan & Richie book *The C Programming Language*
  - MIT open course *Practical Programming in C* (linked off webpage)
  - Prof. Drew Hilton's video tutorials (linked off webpage)

Programming on Linux Machines

- **Remote Access**
  - We will use Duke OIT Linux machines for portions of this course. Read this document about remote access to these machines.
    - » http://people.duke.edu/~bcl15/teachdir/ece250_fall15/remoteaccess.pdf
- **Linux Tutorial**
  - You should also go through this short tutorial on Linux.
    - » http://www.cs.duke.edu/~alvy/courses/unixtut



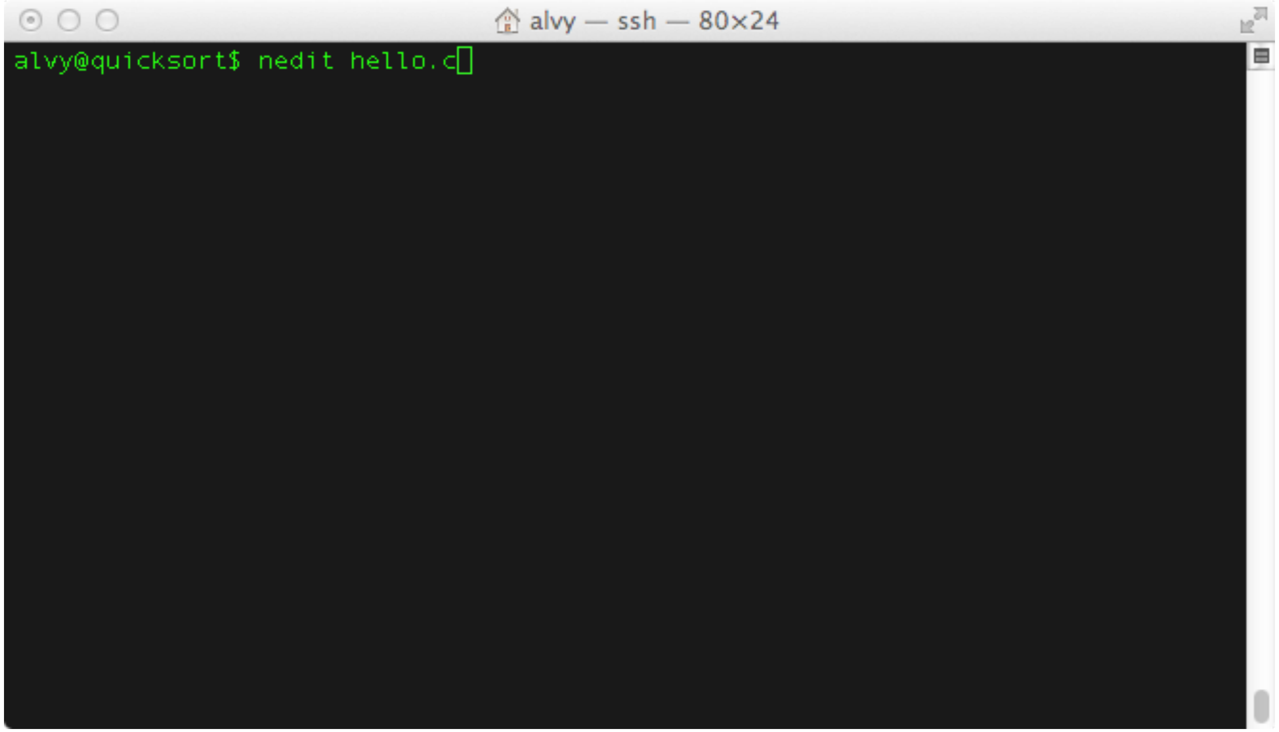
Docs/Resources

Remote access to Linux machines.
We will use the Duke OIT Linux machines for portions of this course.
Please read this document on how to remotely access these machines.

Unix Tutorial
You should also go through this short tutorial on Linux

Creating a C source file

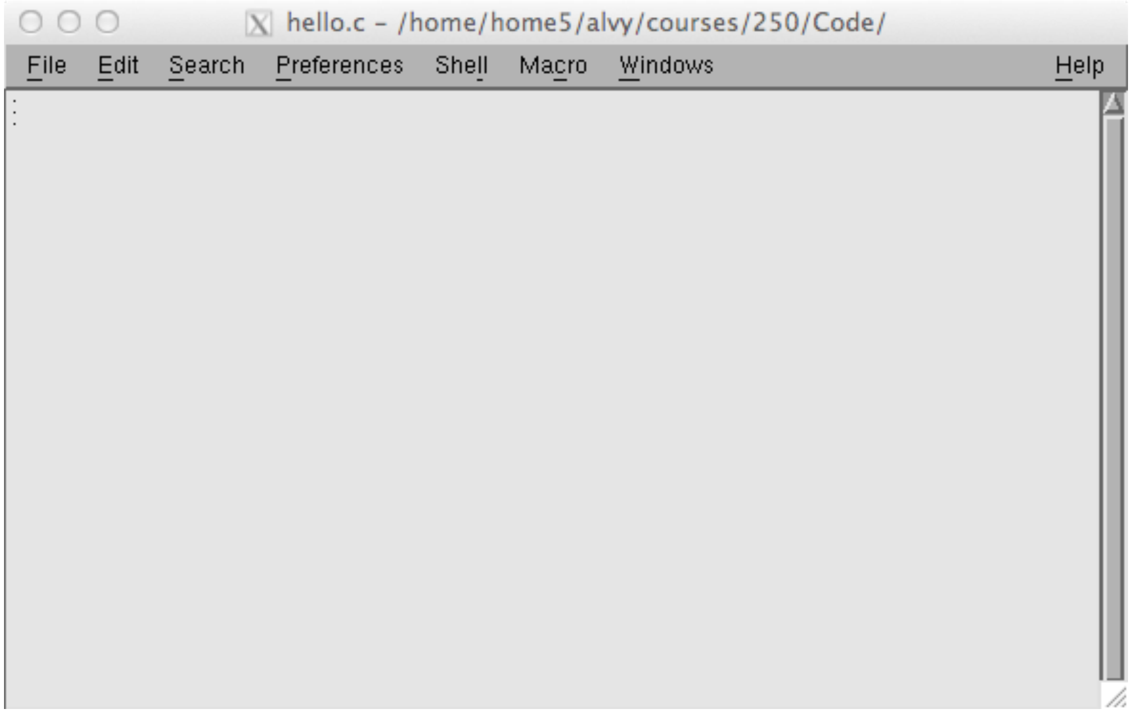
- We are not using a development environment (IDE)
- You will create programs starting with an empty file!
- Files should use .c file extension (e.g., hello.c)
- On a linux machine, edit files with nedit (or emacs or …)



```
alvy@quicksort$ nedit hello.c
```

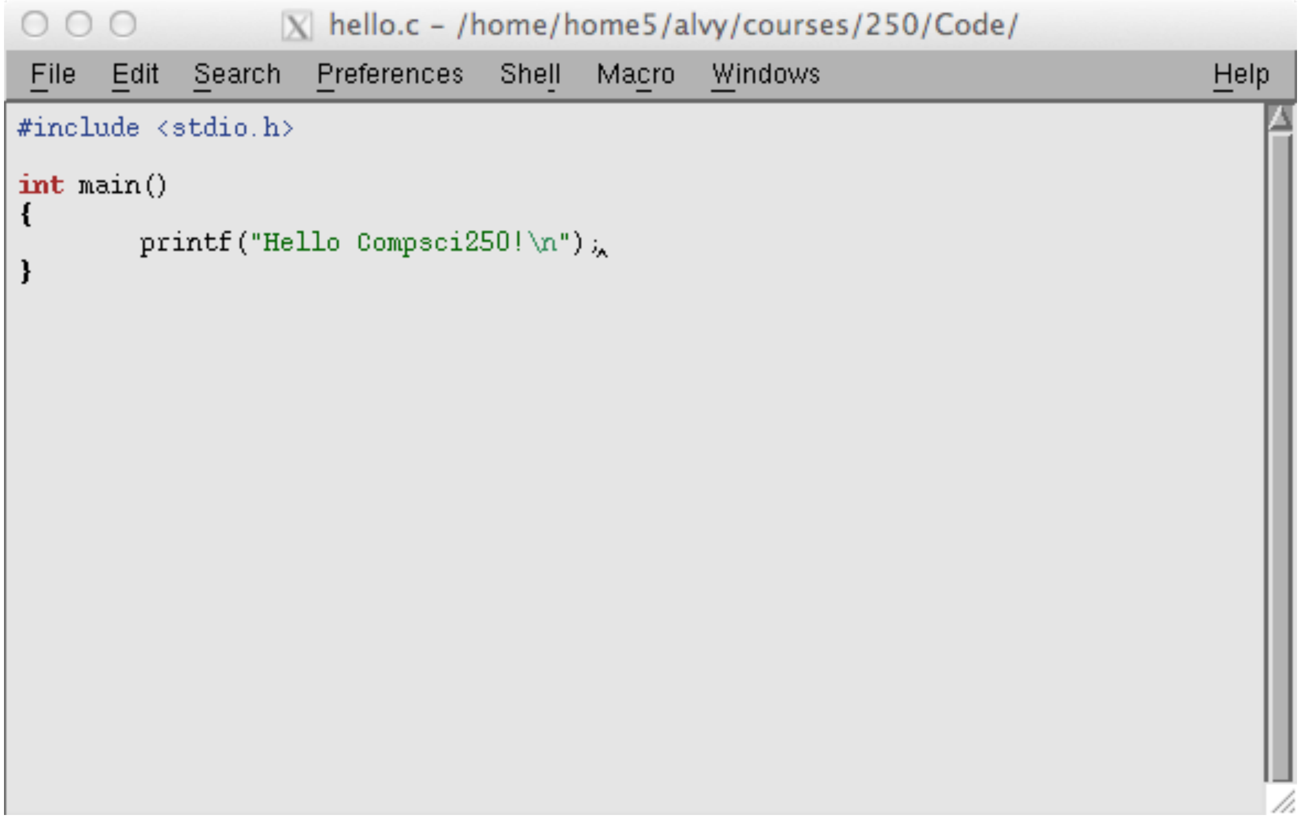
The nedit window

- 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

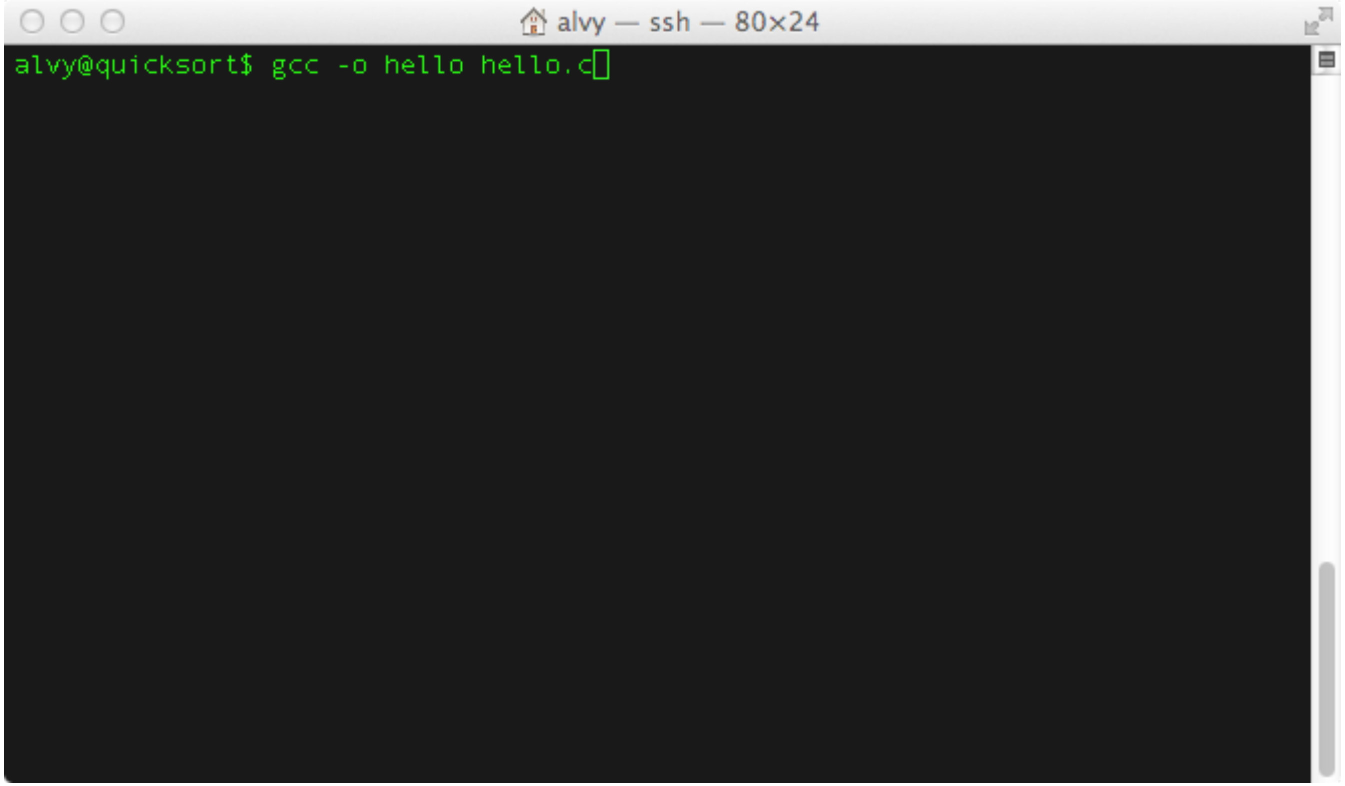


```c
#include <stdio.h>

int main()
{
        printf("Hello Compsci250!\n");
}
```

Compiling the Program

- Use the gcc (or g++) compiler to turn .c file into executable file
- gcc –o <outputname> <source file name>
- gcc –o hello hello.c (you must be in same directory as hello.c)
- If no –o option (i.e., gcc hello.c), then default output name is a.out

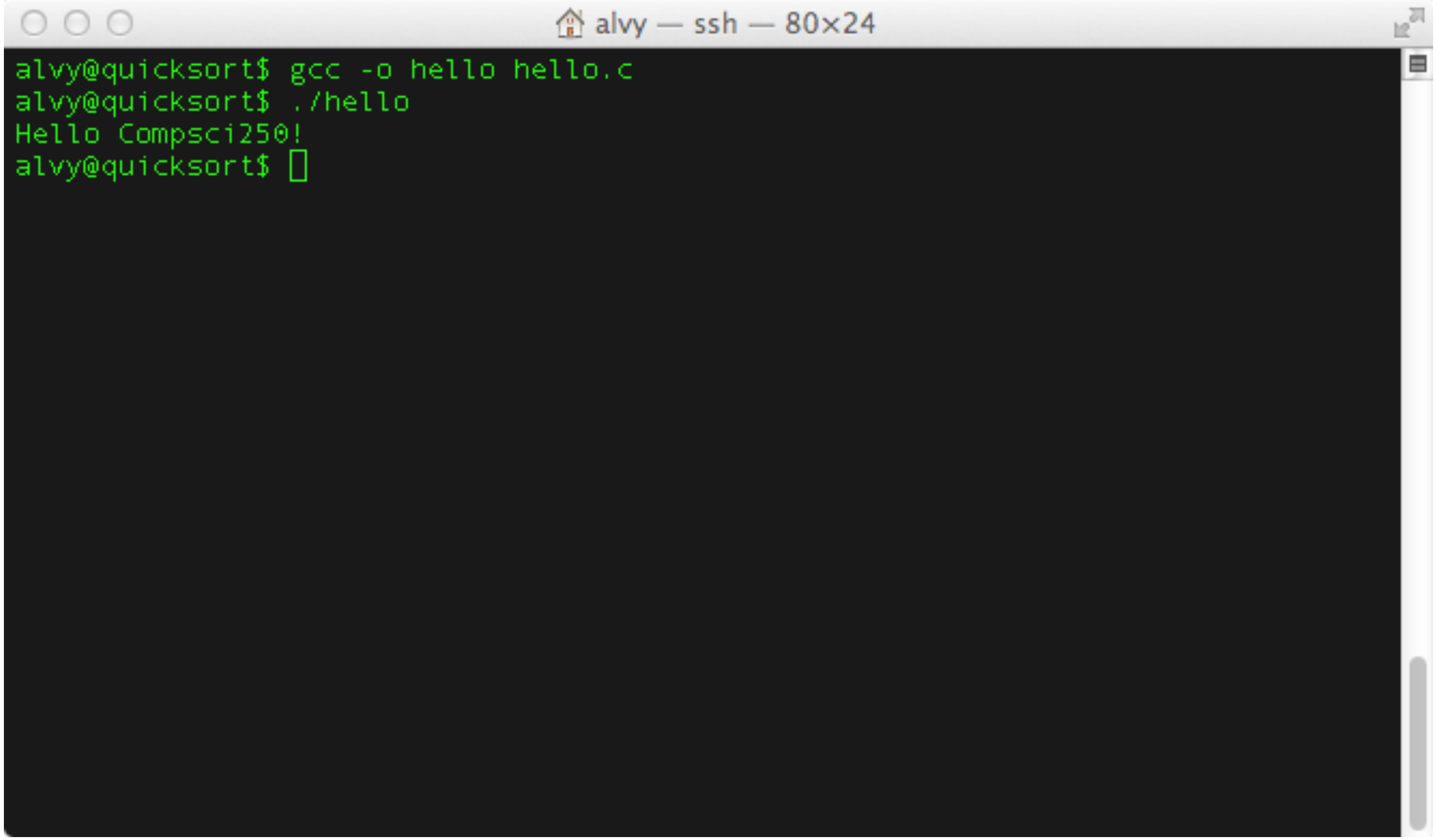


```
alvy@quicksort$ gcc -o hello hello.c
```

© Daniel J. Sorin
From Hilton, Lebeck, Lee, Roth

Running the Program

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



```
alvy@quicksort$ gcc -o hello hello.c
alvy@quicksort$ ./hello
Hello Compsci250!
alvy@quicksort$ 
```

Debugging (where most time is spent)

- OK option #1: “printf debugging”
    - Just print info at different points in the program
    - Not the most efficient approach, but often good enough

- Much better option #2: use a debugger
    - gdb (GNU debugger)
    - Run with: gdb <executable filename>
    - If you get good at using a debugger it is easier/better than printf debugging…
    - Good for stopping at set points in program, inspecting variable values.

Variables, operators, expressions – just like Java

Same as Java!

- Variables types
  - 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
- Operators
  - Mathematical +, -, *, /, %,
  - Logical !, &&, ||, ==, !=, <, >, <=, >=
  - Bitwise &, |, ~, ^ , <<, >> (we'll get to what these do later)
- Expressions: var1 = var2 + var3;

Arrays – same as Java

Same as Java (for now…)

- char buf[256];
- int grid[256][512]; /* two dimensional array */
- float scores[4196];
- double speed[100];

```
for (i = 0; i< 25; i++)
      buf[i] = ‘A’+i;      /* what does this do?  */
```

Strings – not quite like Java

- 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?

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

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

Structures

- Structures are sort of like Java objects
  - They have member variables
  - But they 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;
```

© Daniel J. Sorin
From Hilton, Lebeck, Lee, Roth

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;
}
```

C Allows Type Conversion with Casts

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

```
main() {
        float x;
        int i;
        x = 3.6;
        i = (int) x;  // i is the integer cast of x
        printf(“x=%f, i=%d”, x, i)
}
result: x=3.600000, i=3
```

Variable Scope: Global Variables

- Global variables are accessible from any function
- Declared outside main()

```
#include <stdio.h>
int X = 0;
float Y = 0.0;
void setX() { X = 78; }
int main()
{
        X = 23;
        Y =0.31234;
        setX();
        // what is the value of X here?
}
```

- What if we had “int X = 23;” in main()?

© Daniel J. Sorin

Control Flow – just like Java

Same as Java!

- Conditionals

```
if (a < b) { … } else {…}
switch (a) {
        case 0: s0; break;
        case 1: s1; break;
        case 2: s2; break;
        default: break;
}
```

- Loops

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

Functions – mostly like Java

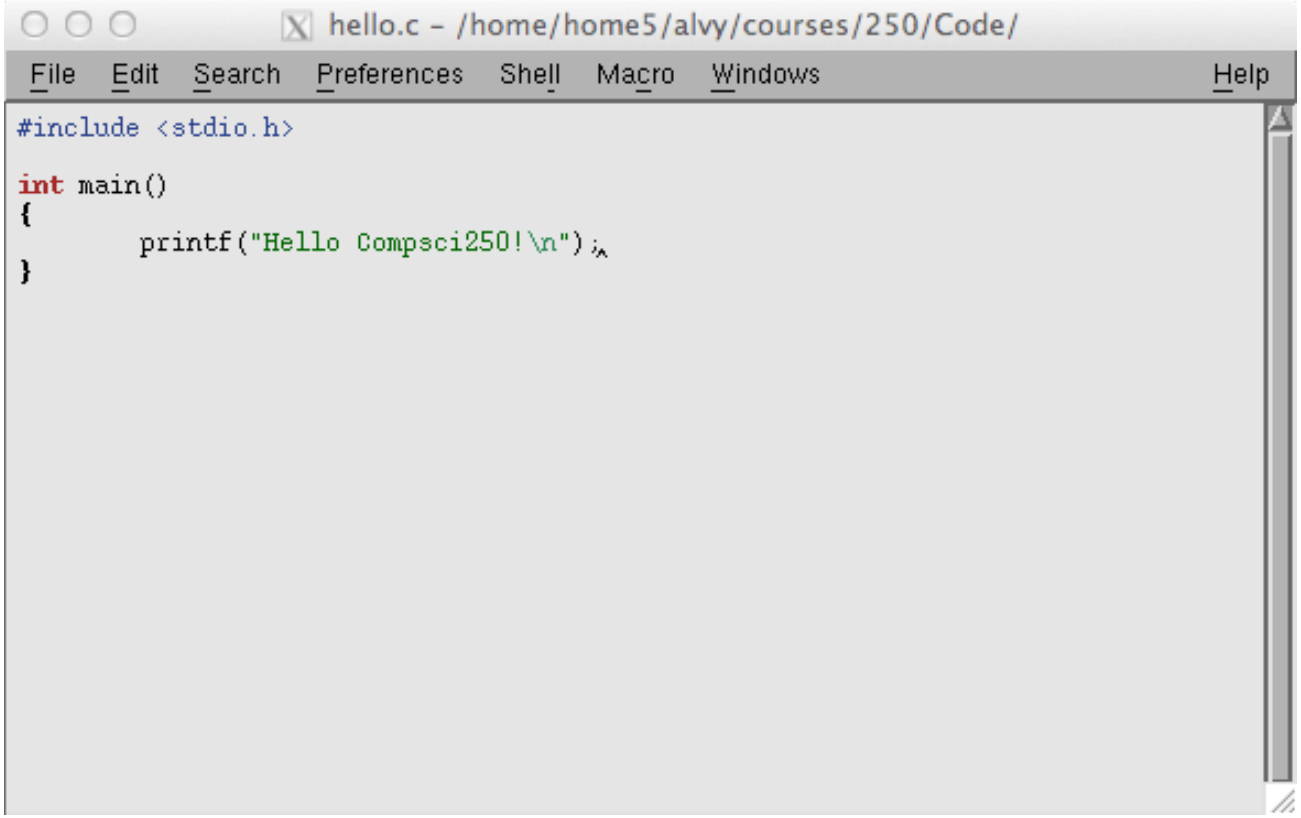
- C has functions, just like Java
  - But these are not methods! (not attached to objects)
- Must be declared before use

```
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 (just like you have libraries in Java)
- printf(args) writes to terminal



```c
#include <stdio.h>

int main()
{
        printf("Hello Compsci250!\n");
}
```

Input/Output (I/O)

- 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 I/O 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)

Formatted I/O

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

printf() = print formatted
scanf() = scan (read) formatted

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

Example: 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 can 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

Header Files, Separate Compilation, Libraries

- C pre-processor provides useful features
  - #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 (functionally equivalent to const int)
- 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
- Libraries: Collection of common functions (some provided, you can build your own)
    - » We've already seen stdio.h for I/O
    - » libc has I/O, 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

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
- Example: myProgram ben 250
  - argc=3
  - argv[0] = “myProgram”, argv[1]=“ben”, argv[2]=“250”

```
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]);
}
```

The Big Differences Between C and Java

1) Java is object-oriented, while C is not

2) Memory management

All variables live in memory (much more on this later!)

- Java: the virtual machine worries about where the variables "live" and how to allocate memory for them
- C: the programmer does all of this

Everything else is approximately the same!

Yes, there are differences, but they're minor

Let's delve into memory management now …

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 [insert evil cackle here]

Pointers

- Declaration of pointer variables (yes, pointers are vars!)
  - int* x_ptr;   // int* is a type – it's a pointer to an int
  - char* c_ptr; // char* is not the same type as int*
  - void* ptr;   // don't ask   ☺

- How do we get the location (address) of a variable?
  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 value from call to memory allocator malloc()
     - » x_ptr = (int*) malloc(sizeof(int));

- Much more about addresses and pointers later, after we learn more about memory

Pointers

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

	statement	x	x_ptr
1	int x;	??	??
2	int *x_ptr;	??	??
3	x = 2	2	??
4	x_ptr = &x;	2	&x
5	*x_ptr = 68;		
6	x_ptr = 200;		

Pointers

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

	statement	x	x_ptr
1	int x;	??	??
2	int *x_ptr;	??	??
3	x = 2	2	??
4	x_ptr = &x;	2	&x
5	*x_ptr = 68;	68	&x
6	x_ptr = 200;	68	200
7	*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        p;
}
main()
  int a
  int
  sw       b
  pr     "a =       b= %d
\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);
}
```

© Daniel J. Sorin
From Hilton, Lebeck, Lee, Roth

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 refs== 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 entry {
    int id;
    struct entry* next;
};
main()
{
  struct entry *head, *ptr;
  head=(struct entry*)malloc(sizeof(struct entry));
  head->id = 66;
  head->next = NULL;

  ptr = (struct entry*)malloc(sizeof(struct entry));
  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);
}
```

Summary

- C Language is lower level than Java
- Many things are similar
  - Data types
  - Expressions
  - Control flow
- Two very important differences
  - No objects!
  - Explicit memory management
- Up Next:
  - So what exactly are those chars, ints, floats?
  - And what exactly is an address?

Resources

- MIT Open Course
- Video snippets by Prof. Drew Hilton (Duke ECE/CS)
  - Doesn’t work with Firefox (use Safari or Chrome)

© Daniel J. Sorin

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 addresses?