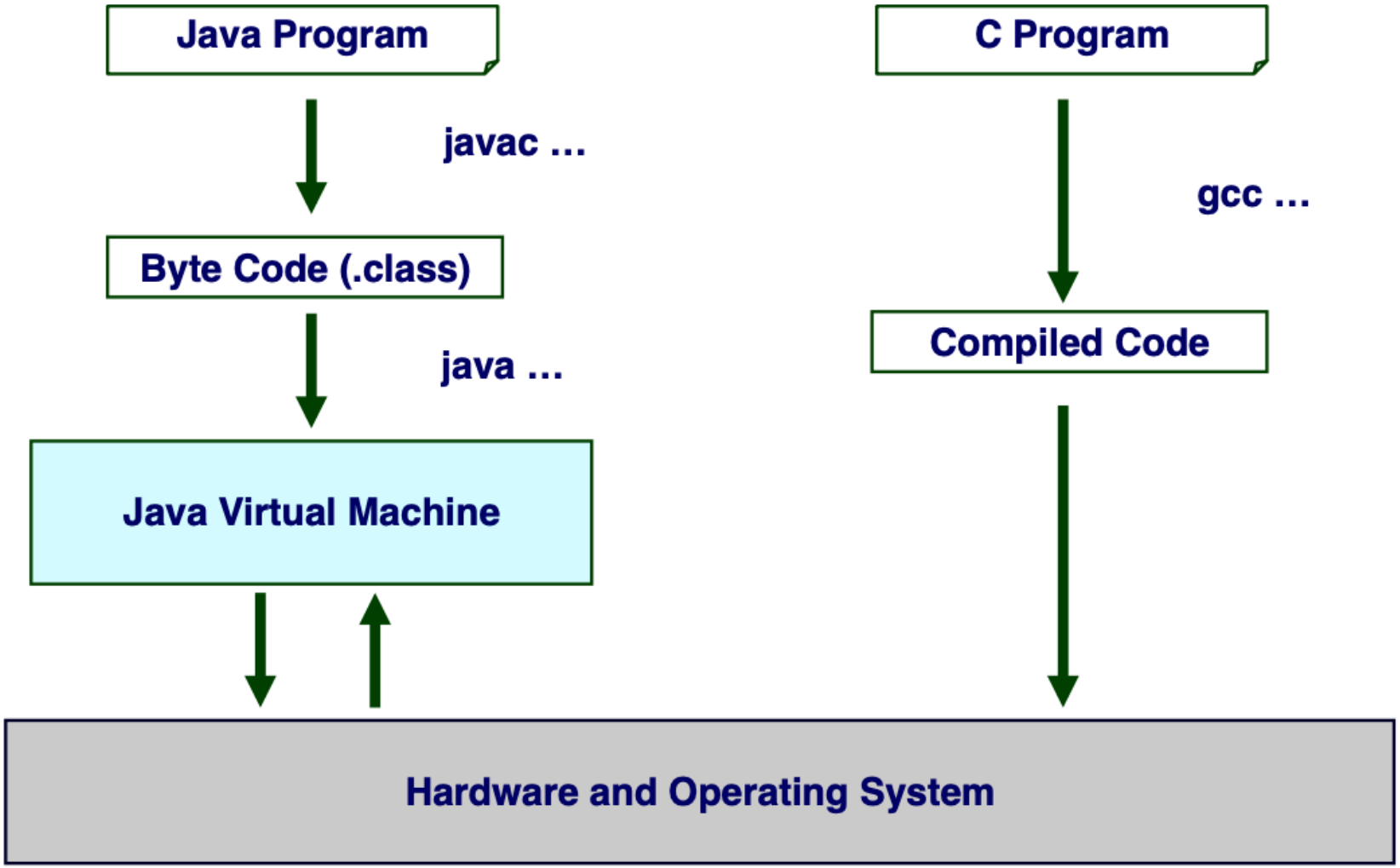
Introduction to C Programming

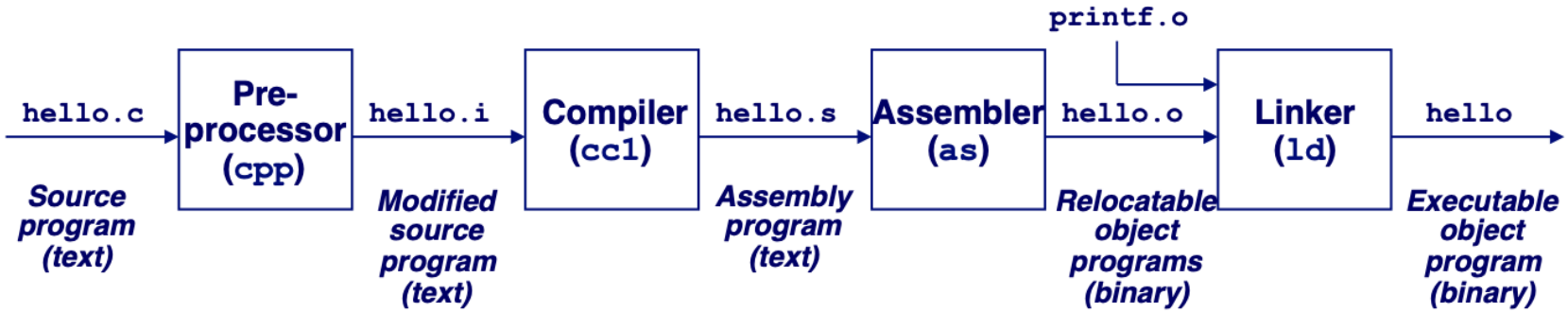
Why Learn C?

- You are learning to be a computer scientist
  - Languages are just tools
  - Choose the tools appropriate for the task
- Current task: Learn how programs written in high-level languages run on computers
- C is closer to machine compared to higher level languages such as Java

Comparison with Java



C Program to Machine Execution



- Compilation System Phases:
  1. Preprocessing: process directives
  2. Compilation: translated into assembly code
  3. Assembly: translates into machine language
  4. Linking: merges libraries

C vs Java Comparison

C

- Procedural
- Compiled
- Low-Level
- User manages memory

Java

- Object Oriented
- Interpreted
- High-Level
- Garbage collector

Anatomy of a C Program

```
#include <stdio.h>
#include <stdlib.h>

char cMessage[] = "Hello\n";

/* Execution will start here */
int main (int argc, char **argv)
{
    int i, count;

    count = atoi(argv[1]);
    for (i = 0; i < count; i++) {
            printf("Hello %d\n", i);
    }
}
```

include files

declaration of global variables

comment

one or more function; each program starts execution at “main”

declaration of local variables

code implementing function

Commenting

- Two ways to comment
- Start comment with //
  - comments until end of line
  - Example:

```
// This is a comment
// This is a second comment
```

- Start comment with /* and end with */
  - comments can span multiple lines
  - Example:

```
/*  This is a comment
that spans multiple lines */
```

- Comments are critical for good development

Variable Declarations

- Each variable has a type, which tells how the compiler how the data is to be interpreted (and how much space it needs, etc)
  - Examples:

    ```
    int counter;
    char letter;
    ```

- Variables can be global or local
  - Global : declare outside scope of any function and can be accessed anywhere
  - Local: declare inside the scope of a function, only accessible from inside of the function

Basic Data Types

Keyword	Data Type	Examples
char	individual character	‘a’, ‘b’, ‘\t’, ’\n’
int	integers	-10, 15, 0, 1324
float	real numbers	-24.4, 0, 4.23
double	real numbers with double precision	-24.4, 0, 4.23

- Modifiers
  - short, long : control size/range of numbers
  - signed, unsigned: include negative numbers or not

Arithmetic Operators

Symbol	Operation	Usage
+	addition	X + Y
-	subtraction	X - Y
*	multiplication	X * Y
/	division	X / Y
%	modulus	X % Y

Special Operators

- Changes value of variable before (or after) its value is used in an expression
- ++ increments a variable
- -- decrements a variable
- Can be used before or after a value

Use	Operation
X++	post increment
++X	pre increment
X--	post decrement
--X	pre decrement

- pre: increment/decrement before using its value
- post: increment/decrement after using its value

Relational Operators

Symbol	Operation	Usage
>	greater than	X > Y
>=	greater or equal to	X >= Y
<	less than	X < Y
<=	less thank or equal to	X <= Y
==	equal	X == Y
!=	not equal	X != Y

- Result is 1 (true) or 0 (false)
- Remember not to confuse equality (==) with assignment (=)

Logical Operators

Symbol	Operation	Usage
!	logical NOT	!X
&&	logical AND	X && Y
||	logical OR	X || Y

- Treats entire variable (or value) as True or False
- Result is either 0 (false) or non-zero (true)

Bit-wise Operators

Symbol	Operation	Usage
~	complement	~X
&	bit-wise AND	X & Y
|	bit-wise OR	X | Y
>>	bit-wise shift right	X >> 2
<<	bit-wise shift left	X << 2

- Operate on bits of variables
- For example:
  - ~0101 = 1010
  - 0101 & 1010 = 0000
  - 0101 | 1010 = 1111
  - 0101 >> 2 = 0001
  - 0101 << 3 = 1000

Control Statements

- Conditional
  - if else
  - Switch
- Iteration (loops)
  - for
  - while
  - do while
- Specialized “go-to”
  - break
  - continue

If Else Statements

```
if (expression){
        statement
}
```

- If an expression is true, then run the statement

```
if (expression A){
        statement 1;
} else {
        statement 2;
}
```

- If an expression is true, then run the statement 1, else run another statement 2

```
if (expression A){
        statement 1;
} else if (expression B){
        statement 2;
} else {
        statement 3;
}
```

- Evaluates all expressions until finds one with non-zero result
  - Executes corresponding statements
  - If all expessions evaluate to zero, executes statements for "else" branch

The Switch Statement

```
switch (expression) {
        case const-1:
                statement 1;
        case const-2:
                statement 2;
        case const-3:
                statement 3;
        default:
                statement-n;
}
```

- Evaluates expression and skips to case corresponding the the result.
- Result must be an integer
- Executes statements corresponding to the result and continues executing ultil encountering a break or end of switch statements
- default always matches

Switch Statement (example)

```
int fork;
…
switch (fork) {
      case 1:
            printf(“take left”);
      case 2:
            printf(“take right”);
            break;
      case 3:
            printf(“make U turn”);
            break;
      default:
            printf(“go straight”);
}
```

Iterations (Loops)

Format	Description

for (start-expression;
condition;
update-expression){
…
}
	- runs zero or more times
	- keeps iterating while cond-expression != 0
	- compute start-expr before first iteration
	- compute update-expr after each iteration

while (expression) {
…
}
	- runs zero or more times
	- keeps iterating while expression != 0
	- compute expression before each iteration

do { … } while (expression)	- runs one or more times - keeps iterating while expression != 0 - computes expression after each iteration

Specialized Go-to's

- break;
  - force immediate exit from switch or loop
  - goes to statement immediately following switch/loop
- continue;
  - skip the rest of the computation in the current iteration of the loop
  - goes to evaluation of conditionals expression for execution of the next iteration

Specialized Go-to's (example)

```
int index = 0;
int sum = 0;

while ( (index >= 0) && (index <= 20) ){
    index = index +1;
    if (index == 11){
        break;
    }
    if ( (index % 2 ) == 1){
        continue;
    }
    sum = sum + index;
}
```

Functions

- Similar to Java Method
- Components
  - Return Type
    - void if no return value
  - Function name
  - Parameters
  - Body
    - statements to be executed
    - return forces exit from function, resuming at statement immediately after function call

```
int Factorial(int n) {
        int i;
        int result = 1;
        for (i = 1; i <= n; i++){
                result = result * i;
        }
        return result;
}
```

Function Calls

- Function call as part of an expression
  - Example: z = x + Factorial(y);
  - Arguments evaluated before function call
  - Return value is used to compute expression
  - Cannot have a void return type
- Function Call as a statement
  - Example: Factorial(y);
  - Can have a void return type
  - Returned value is discared (if there is one)

Basic Input and Output

- Variety of I/O functions in C standard library
- Input:
  - int printf(const char *format, ….);
  - Writes to standard output based on the format provided
  - Examples:
    - printf(“Hello world\n”);
    - printf(“My number grade is %d\n”, grade);
- Input:
  - int scanf(const char *format, …);
  - Reads input from standard input and scans it based on formatting provided
  - Examples:
    - scanf(“%d”, &grade);
    - scanf(“%d %c”, &grade, &lettergrade)
  - Note: Don’t worry about the “&” yet

Basic Input and Output

Common format specifiers

Specifier	Type
%c	character
%d	integer
%u	unsigned int
%f	float
%lf	double
%s	string
%p	pointer

Common escape sequences

Escape Sequence	Character
\n	newline
\t	tab
\\	backslash
\’	single quotation
\”	double quotation

Basic C Program (HelloWorld.c)

```
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char** argv){
    printf("Hello World!\n");
    return 0;
}
```

Compiling and Running a C Program

- Use gcc to compile your programs
  - Example: gcc HelloWorld.c
  - Will compile into executable named "a.out"
  - Run the program by running ./a.out
- Use the "-o" flag to specify the output name
  - Example: gcc HelloWorld.c -o HelloWorld
  - Will compile program into executable named "HelloWorld"
  - Run the program by running ./HelloWorld

For Next Lecture

- Start practicing C Programming
  - Connect to iLab Machines
  - Start coding, compiling, and running short C programs
  - Get used to the iLab Machine
- For those who want to get ahead
  - Arrays
  - Pointers
  - Structs