

## Chapter 11 Introduction to Programming in C

Based on slides © McGraw-Hill  
Additional material © 2004/2005 Lewis/Martin  
Modified by Diana Palsetta

CIT 593

## The Course Thus Far...

### We did data representation

- Bits, bytes, integers, floating, characters
- Ultimately, to understand ISA

### We did assembly language programming

- Rather than 1's & 0's, deal in human readable form
- Assembler filled some gap between algorithm level & ISA level
- There are still limitations
  - E.g. accessing a label/variable that is not within the PC range.
  - Programmer still has to deal with that

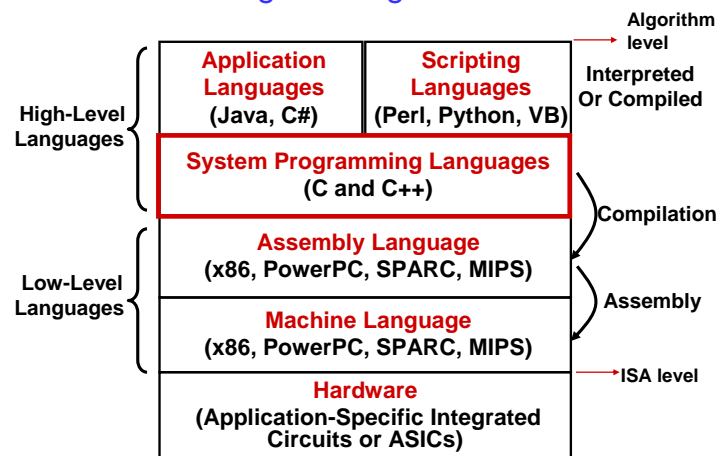
### We will learn High-level language (C programming)

- Basically fill more gap between two levels and make programming easier.
- Ultimately, for a deeper understanding of any language (e.g. Java)

CIT 593

11 - 2

## Programming Levels



CIT 593

11 - 3

## Why High-Level Languages?

### Easier than assembly. Why?

- Less primitive constructs
- Variables that can be of different data types (int, float, char)
- Type checking

### Portability

- Write program once, run it on the LC-3 or Intel's x86
  - Caveat: Not all languages though

### Disadvantages

- Slower and larger programs (in most cases)
- Can't manipulate low-level hardware
  - All operating systems have some assembly in them

**Verdict:** assembly coding is rare today

CIT 593

11 - 4

## Goal

### You already know or are learning Java

- Cover the basics quickly of C language
  - Syntax, programming constructs, data types
- Learn how local, global variables fit in assembly context
- How functions in C are implemented at the machine level
- We'll spend more time on pointers & other C-specific nastiness
  - "Nastiness": memory management in C is really bad

CIT 593

11 - 5

## C to Java Transition

### C and Java were created two decades apart

- C: 1970s - AT&T Bell Labs
- C++: 1980s - AT&T Bell Labs
- Java: 1990s - Sun Microsystems

### Java and C/C++

- Syntactically similar (Java uses C syntax)
- C lacks many of Java's features
- Subtly different semantics

CIT 593

11 - 6

## C is Similar To Java Without:

### Objects

- No classes, objects, methods, or inheritance

### Exceptions

- Check all error codes explicitly
  - That means you do error checking

### Standard class library

- C has only a small standard library

### Garbage collection

- C requires explicit memory allocate and free

### Safety

- Java has strong type checking, checks array bounds
- In C, anything goes

### Portability

- Source: C code is less portable (but better than assembly)

CIT 593

11 - 7

## More C vs Java differences

### Include vs Import

- Java has `import java.io.*;`
- C has: `#include <stdio.h>`

### Boolean type

- Java has an explicit boolean type
- C just uses an "int" as zero or non-zero
- C's lack of boolean causes all sorts of trouble

More differences as we go along...

CIT 593

11 - 8

## Achieving Machine Independence with C

C was initially used as systems programming language due to its low-level capabilities

- Compilers and operating system
- E.g. Unix O.S.

To encourage **machine-independent** programming

- ANSI C is used to standardize the language
- A standards-compliant C program can be compiled for a very wide variety of computer platforms and operating systems with **minimal change** to its source code
- The language has become available on a very wide range of platforms.

CIT 593

11 - 9

## What is C++?

**C++ is an extension of C**

- Backward compatible (good and bad)
- That is, all C programs are legal C++ programs

**C++ adds many features to C**

- Classes, objects, inheritance
- Templates for polymorphism
- A larger class library
- Exceptions (not actually implemented for a long time)
- More safety (though still unsafe)
- Operator and function overloading

**Thus, many people use it (to some extent)**

- However, we're focusing on only C, not C++

CIT 593

11 - 10

## Compilation vs. Interpretation

Different ways of translating high-level languages

### Interpretation

- Interpreter: program that executes program statements
  - Directly interprets program (portable but slow)
  - Limited optimization
- Easy to debug, make changes, view intermediate results
- Languages: BASIC, LISP, Perl, Python, Matlab

### Compilation

- Compiler: translates statements into machine language
  - Creates executable program (non-portable, but fast)
  - Performs optimization over multiple statements
- Harder to debug, change requires recompilation
- Languages: C, C++, Fortran, Pascal

### Hybrid

- Java, has features of both interpreted and compiled languages

CIT 593

11 - 11

## Compilation vs. Interpretation

Consider the following algorithm:

```
Get W from the keyboard.  
X = W + W  
Y = X + X  
Z = Y + Y  
Print Z to screen
```

If interpreting, how many arithmetic operations occur?

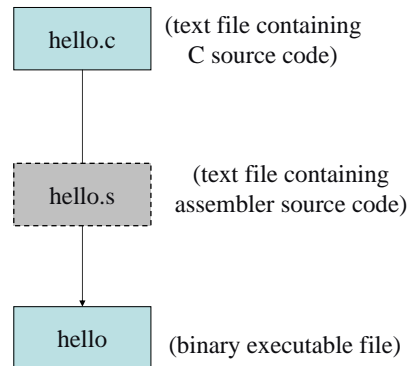
If compiling, we can analyze the entire program and possibly reduce the number of operations.

- Can we simplify the above algorithm to use a single arithmetic operation?

CIT 593

11 - 12

## C Programs are Compiled



CIT 593

11 - 13

## Compilation Process

Entire mechanism is usually called the **compiler**

### Preprocessor

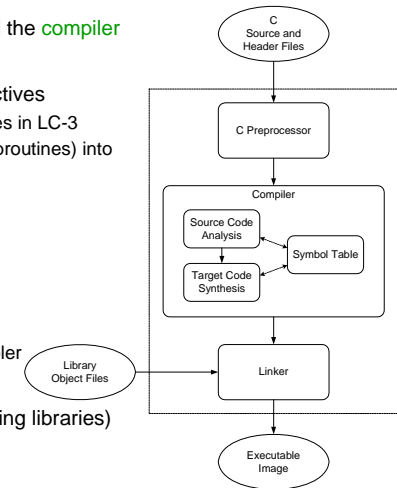
- Acts upon C preprocessor directives
  - These are like pseudo directives in LC-3
  - Inserts contents of file (e.g. subroutines) into the current source file
- “Source-level” transformations
  - Output is still C

### Compiler

- Generates object file
  - Machine instructions (binaries)
  - Similar processor LC3 Assembler

### Linker

- Combine object files (including libraries) into executable image



CIT 593

11 - 14

## Compiler

### 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
- Very dependent on target machine

### Example Compiler: GCC

- The Free-Software Foundation’s compiler
- Many front ends: C, C++, Fortran, Java
- Many back ends: Intel x86, PowerPC, SPARC, MIPS, Itanium

CIT 593

11 - 15

## C Preprocessor Directives

### #include <stdio.h>

- Before compiling, copy contents of stdio.h into source code.
- .h files typically contain descriptions of functions and variables needed by the program.

### #define STOP 0

- Before compiling, replace all instances of the string “STOP” in the code with the string “0”
- Called a *macro*
- Used for values that won’t change during execution, but might change if the program is reused. (Must recompile.)

CIT 593

11 - 16

## Comments in C

Begins with `/*` and ends with `*/`

- Can span multiple lines
- Comments are not recognized within a string
  - example: `"my/*don't print this*/string"`  
would be printed as: `my/*don't print this*/string`

Begins with `//` and ends with "end of line"

- Single-line comment e.g. `//This is a comment`
- Much like `;` in LC-3 assembly
- Introduced in C++, later back-ported to C

As before, use comments to help reader.

CIT 593

11 - 17

## main Function

Every C program must have a function called `main()`

- Starting point for every program
- All C programs start and finish execution in the main function
- Similar to Java's main method
  - `public static void main(String[] args)`

The code for the function lives within brackets:

```
int main()           ANSI standard – main() returns an int value
{                   Used for exiting the program:
/* code goes here */ 0 - no error
}                   1 - error
```

Note: `int main(int argc, char **argv)` – main function can also take arguments like in Java

CIT 593

11 - 18

## Variable Declarations

Variables are used as names for data items

Each variable has a *type*, tells the compiler:

- How the data is to be interpreted
- How much space it needs, etc.

```
int counter;
int startPoint;
```

C has similar primitive types as Java

- int, char, long, float, double
- More later

CIT 593

11 - 19

## Input and Output

**Variety of I/O functions in C Standard Library**

- Must include `<stdio.h>` to use them

**Printf – Print formatted**

- Performs output to standard output device
- String contains characters to print and formatting directions for variable

```
printf("%d\n", counter);
```

- This call says to print the variable `counter` as a decimal integer, followed by a newline (`\n`)
- Newline will make the cursor go onto next line

**Scanf – Scan Formatted**

- String contains formatting directions for reading input

```
scanf("%d", &startPoint);
```

- This call says to read a decimal integer and assign it to the variable `startPoint` (`&` - specifies the address of the operand....more on this when we do pointers)

CIT 593

11 - 20

## More About Output

Can print arbitrary expressions, not just variables

```
printf("%d\n", startPoint - counter);
```

Print multiple expressions with a single statement

```
printf("%d %d\n", counter, startPoint - counter);
```

Different formatting options:

- `%d` decimal integer
- `%x` hexadecimal integer
- `%c` ASCII character
- `%f` floating-point number

CIT 593

11 - 21

## Examples

This code:

```
printf("%d is a prime number.\n", 43);  
printf("43 plus 59 in decimal is %d.\n", 43+59);  
printf("43 plus 59 in hex is %x.\n", 43+59);  
printf("43 plus 59 as a character is %c.\n", 43+59);
```

produces this output:

```
43 is a prime number.  
43 plus 59 in decimal is 102.  
43 plus 59 in hex is 66.  
43 plus 59 as a character is f.
```

CIT 593

11 - 22

## Examples of Input

Many of the same formatting characters are available for user input

```
scanf("%c", &nextChar);
```

- reads a single character and stores it in nextChar

```
scanf("%f", &radius);
```

- reads a floating point number and stores it in radius

```
scanf("%d %d", &length, &width);
```

- reads two decimal integers (separated by whitespace), stores the first one in length and the second in width

CIT 593

11 - 23

## A Simple C Program

```
#include <stdio.h>  
#define STOP 0  
  
int main()  
{  
    /* variable declarations */  
    int counter; /* an integer to hold count values */  
    int startPoint; /* starting point for countdown */  
  
    /* prompt user for input */  
    printf("Enter a positive number: ");  
    scanf("%d", &startPoint); /* read into startPoint */  
  
    /* count down and print count */  
    for (counter=startPoint; counter >= STOP; counter--) {  
        printf("%d\n", counter);  
    }  
}
```

CIT 593

11 - 24

## Remaining Chapters

A more detailed look at many C features

- Variables and declarations
- Operators
- Control Structures
- Functions
- Pointers and Data Structures
- I/O

Emphasis on how C is converted to assembly language

### C help

- Also see "C Reference" in Appendix D
- Use online resources on Technical/Programming Link on CIT 593 website
- Famous Book: The C Programming Language by B Kernighan and D. Ritchie (on Library Reserve)