Introduction to Cybersecurity
A Software/Hardware Approach

Brief Computer History & C Introduction

Prof. Michel A. Kinsy

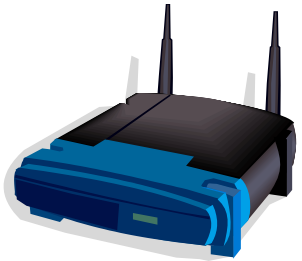
BOSTON UNIVERSITY
Department of Electrical & Computer Engineering

Computing Devices Now



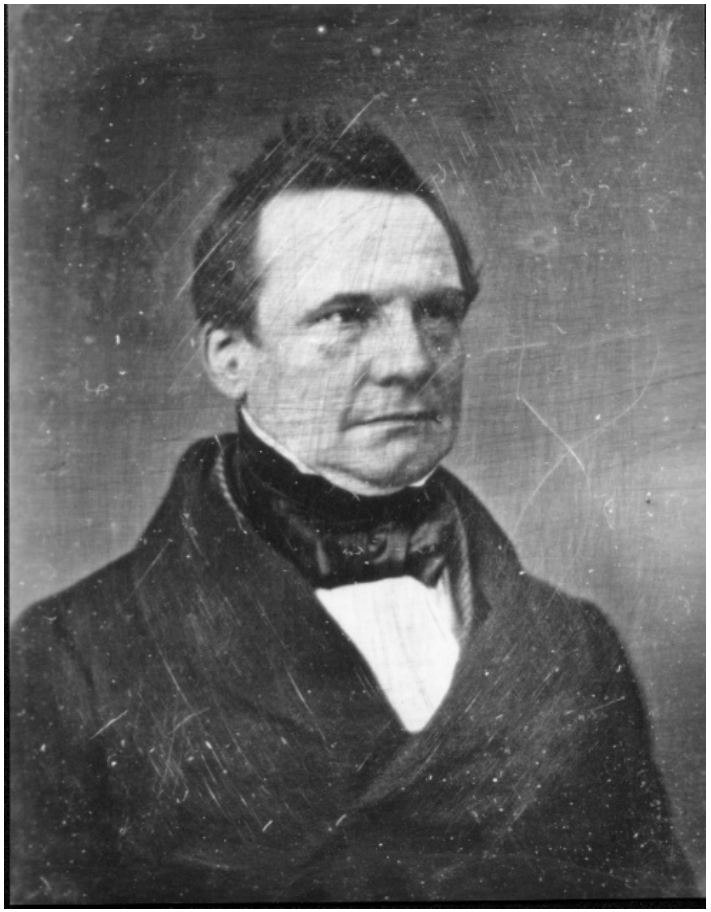






Charles Babbage 1791-1871

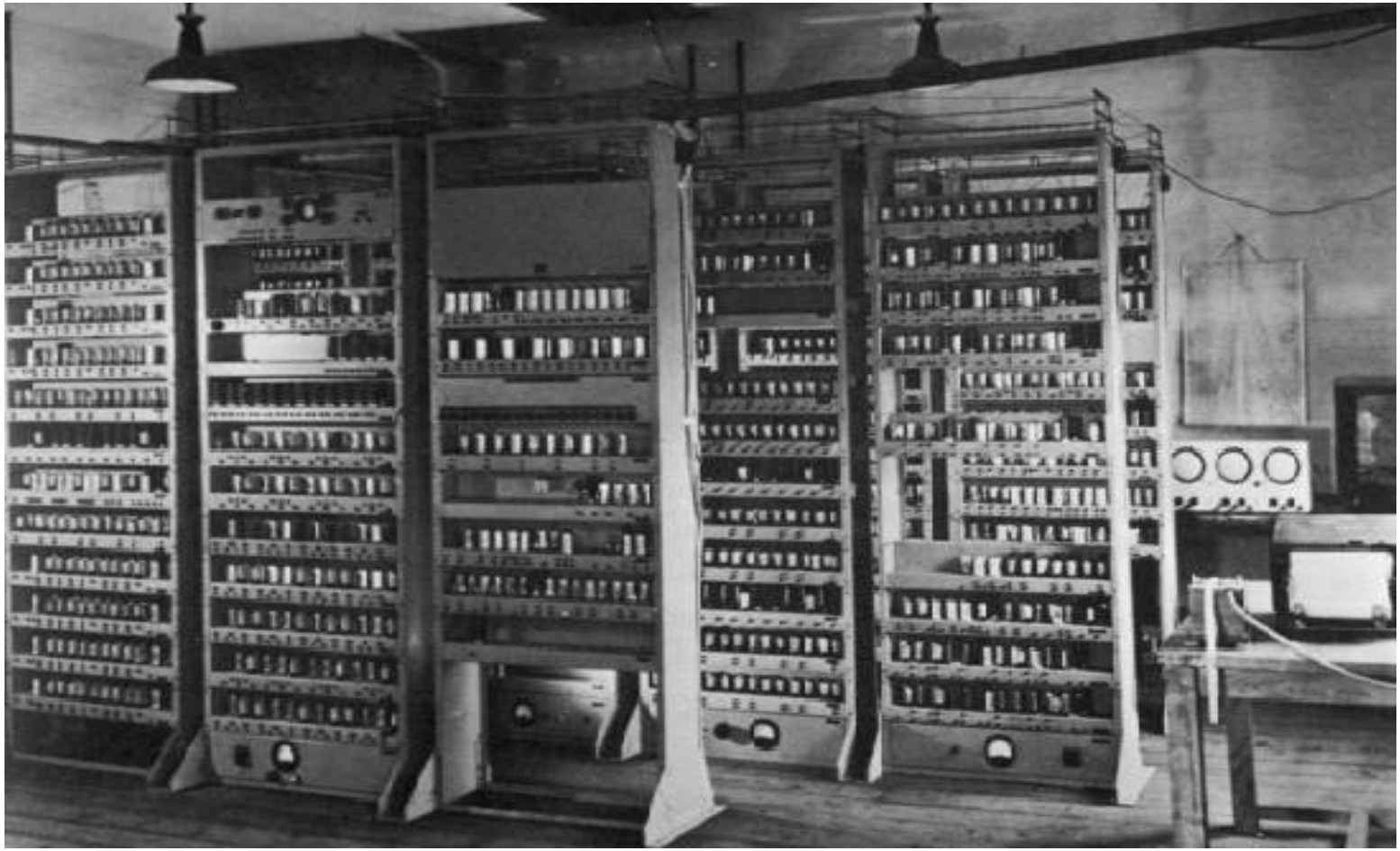


- Difference Engine 1823
- Analytic Engine 1833
  - The forerunner of modern digital computer!
    - Application
      - Mathematical Tables – Astronomy
      - Nautical Tables – Navy
    - Background
      - Any continuous function can be approximated by a polynomial
      - Any Polynomial can be computed from difference tables

The First Programmer

- Ada Byron aka "Lady Lovelace" 1815-52
  - Ada's tutor was Babbage himself!

Computing Devices Then…



Automatic Computer

- Electronic Discrete Variable Automatic Computer
- ENIAC's programming system was external
  - Sequences of instructions were executed independently of the results of the calculation
  - Human intervention required to take instructions "out of order"
- EDVAC was designed by Eckert, Mauchly and von Neumann in 1944 to solve this problem
  - Solution was the stored program computer
  - "program can be manipulated as data"

The Big Idea in Today's Computers

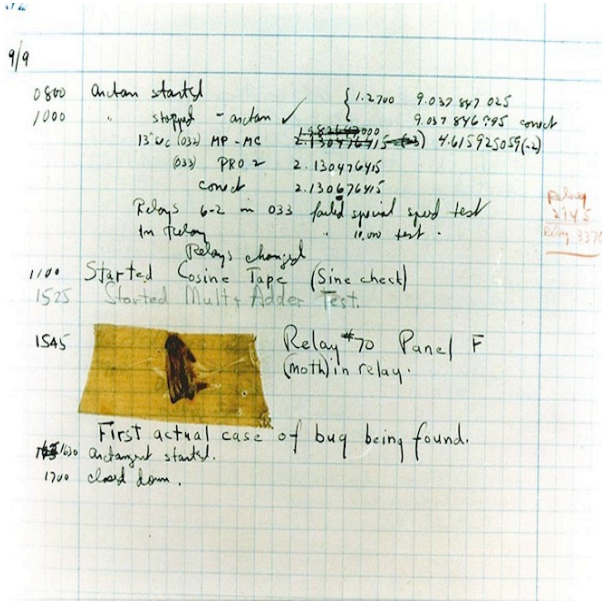
- Stored Program Computer

Program = A sequence of instructions

- How to control instruction sequencing?
  - Manual control
    - Calculators
  - Automatic control external (paper tape)
    - Harvard Mark I , 1944
    - Zuse's Z1, WW2
  - Internal
    - Plug board ENIAC 1946

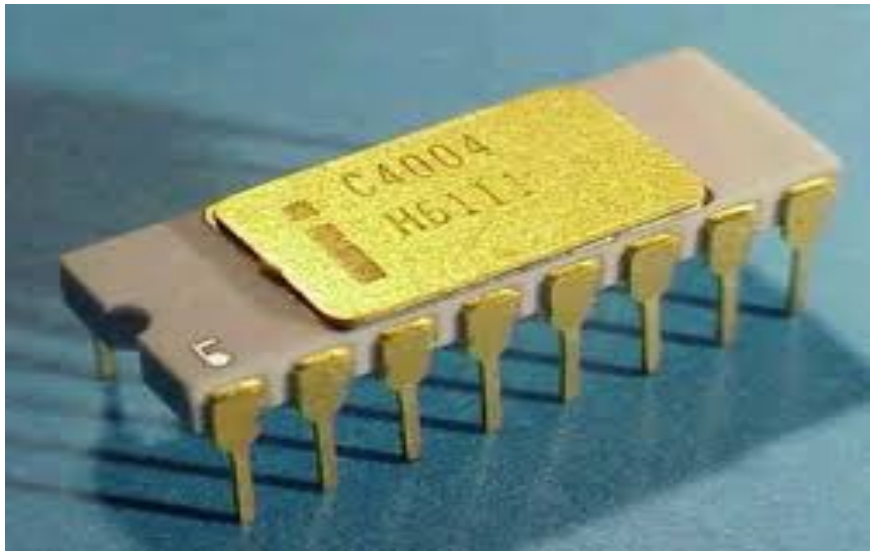
First Program Bug

- The first computer bug is a moth!
- Grace Murray Hopper found the bug while working on the Harvard Mark II computer

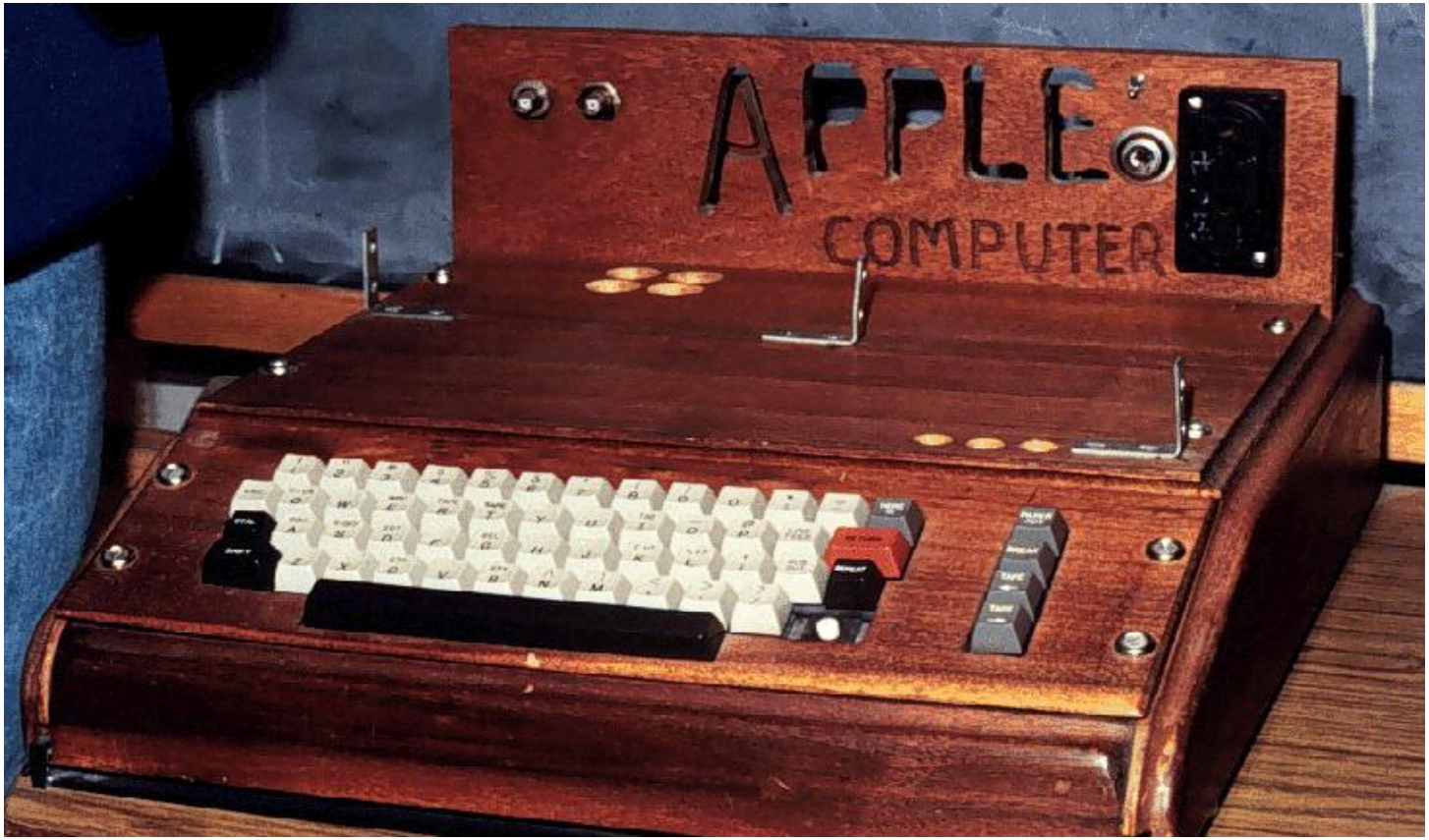


First Microprocessor

- By Intel Corporation
  - 4-bit Microprocessor 4004 in 1971
  - 8-bit microprocessor 8008 in 1972



Apple 1 Computer - 1976



IBM PC - 1981

- IBM-Intel-Microsoft joint venture
  - First wide-selling personal computer used in business
  - 8088 Microchip - 29,000 transistors
  - 4.77 Mhz processing speed
  - 256 K RAM (Random Access Memory) standard

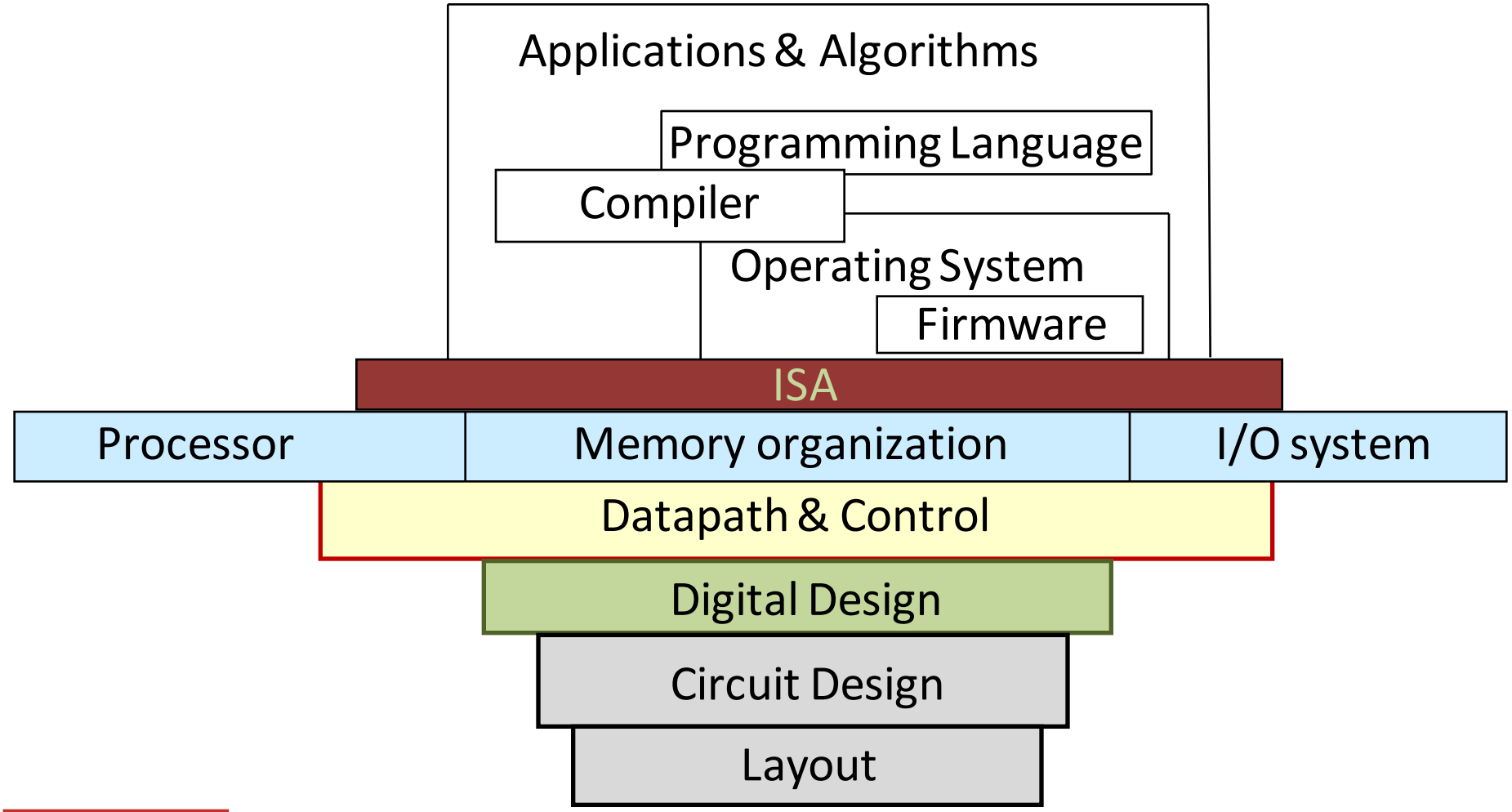
Apple Macintosh - 1984

The Amiga 1000 1985

PowerPC 1991

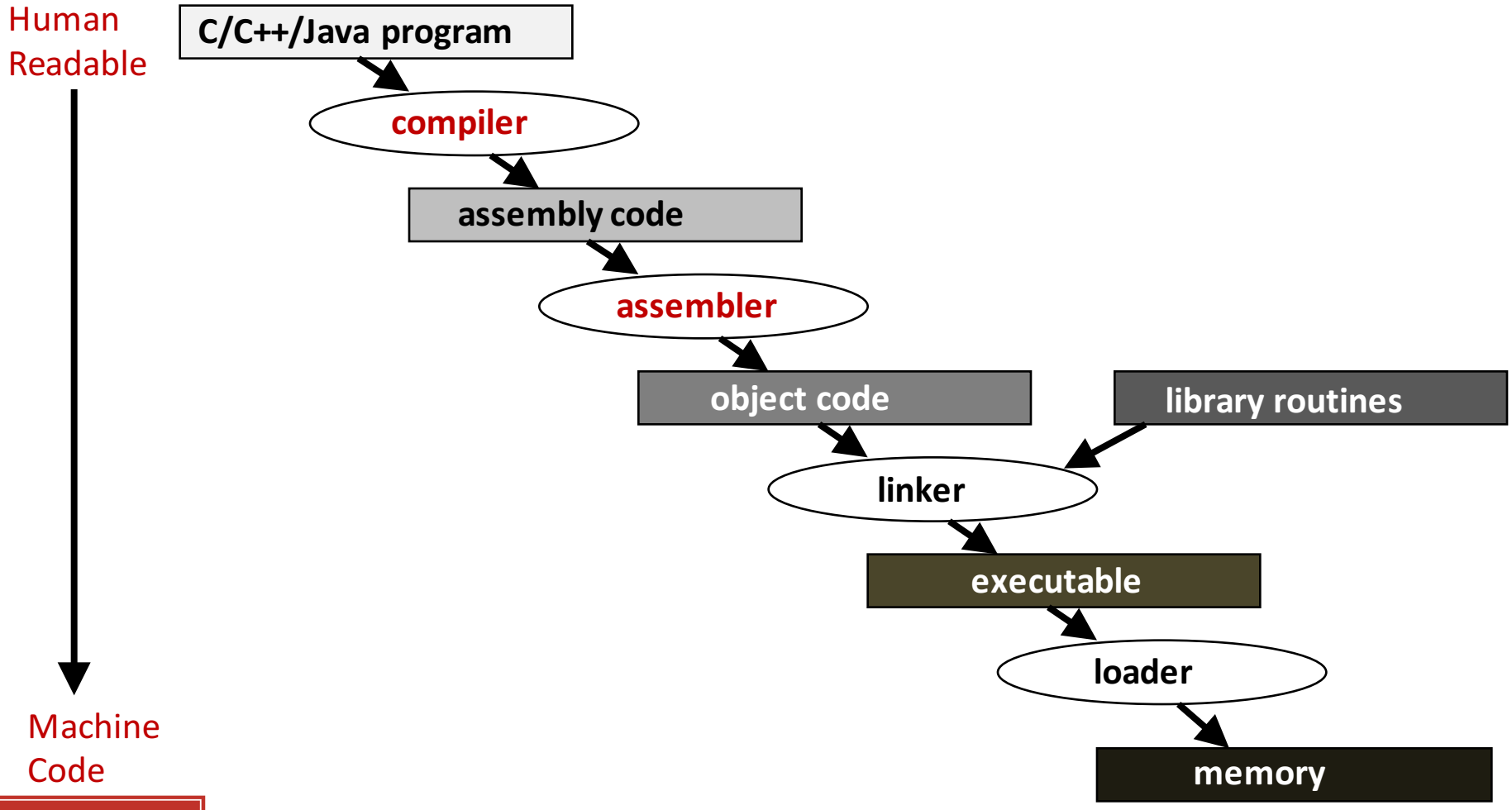
Apple 2016

The Computing Stack



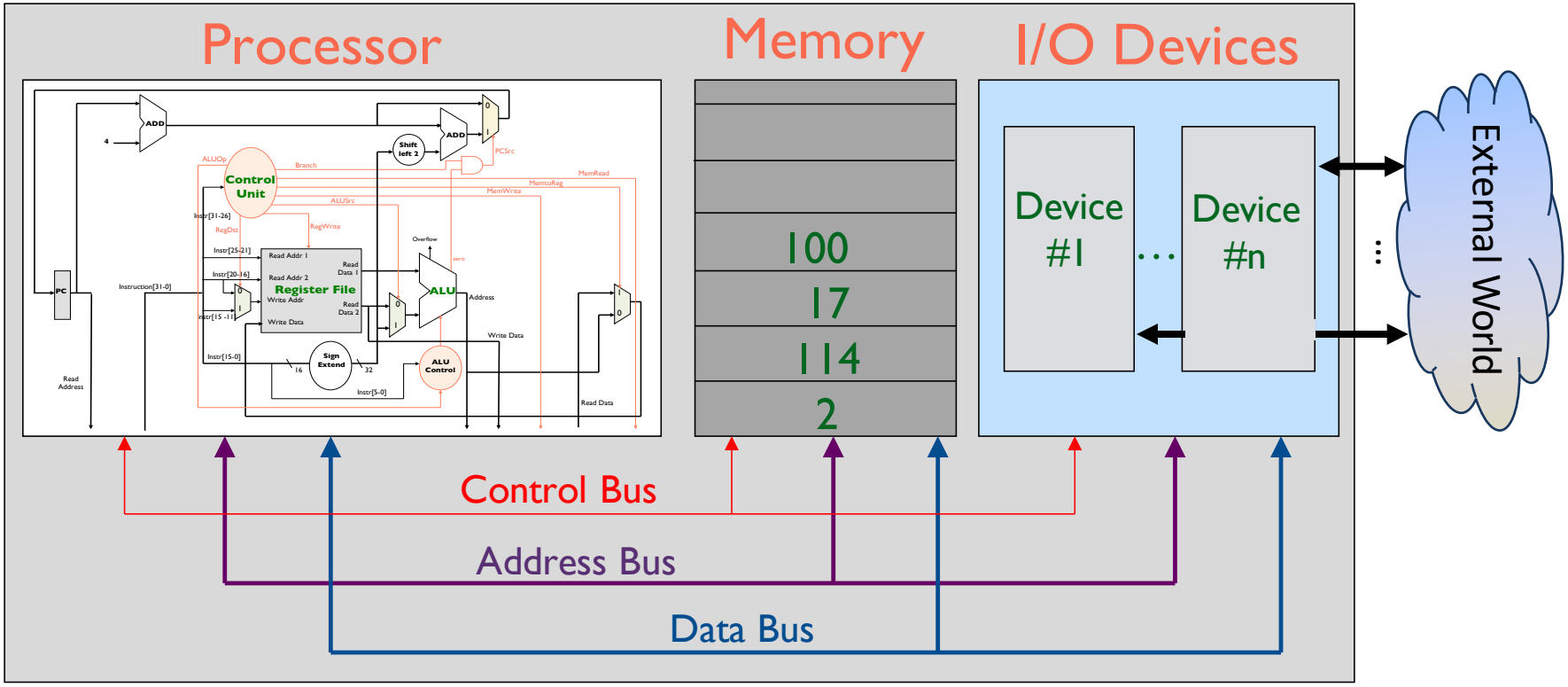
Bridging/Compiling Process

- High-Level Language



The Overall Organization!

- The modern computer system has three major functional hardware units: CPU (Processing Engine), Main Memory (Storage) and Input/Output (I/O) Units



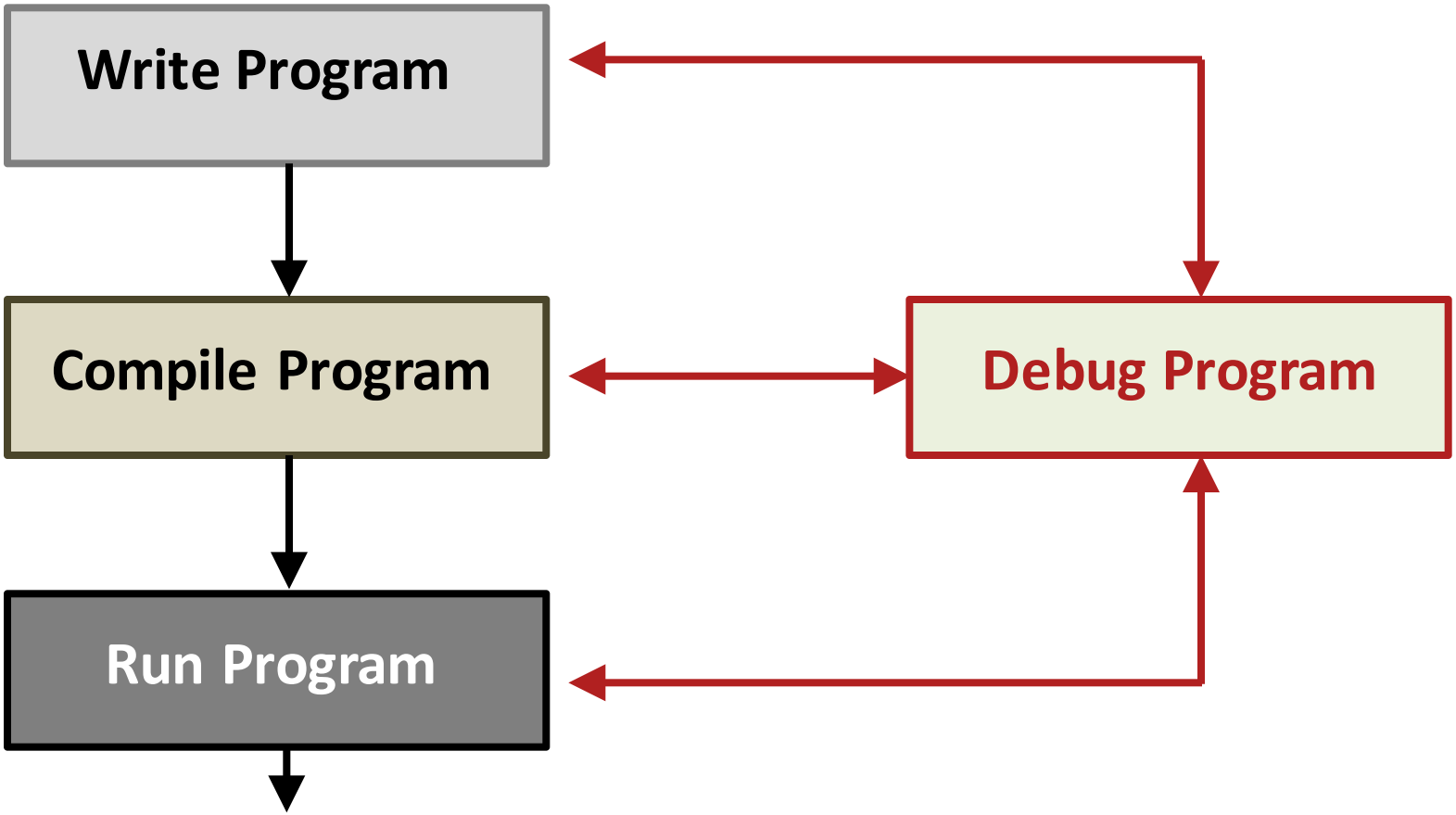
Programming Languages

- There are many programming languages available: Pascal, C, C++, Java, Ada, Perl and Python
- All of these languages share core concepts
- By focusing on these concepts, you are better able to learn any programming language

Programming Languages

- There are many programming languages available: Pascal, C, C++, Java, Ada, Perl and Python
- All of these languages share core concepts
- Hence, by learning C, you are poised to learn other languages, such as Java or Python
  - In this class, we will learn core programming concepts through the powerful C language
  - Why C? It runs nearly as fast as assembly language code

Programming Process



Introduction to C

- Developed in 1972 by Dennis Ritchie at Bell Labs
- It is imperative programming language
- It provides:
  - Efficiency, high performance and high quality software
  - Flexibility and power
  - Many high-level and low-level operations

Introduction to C

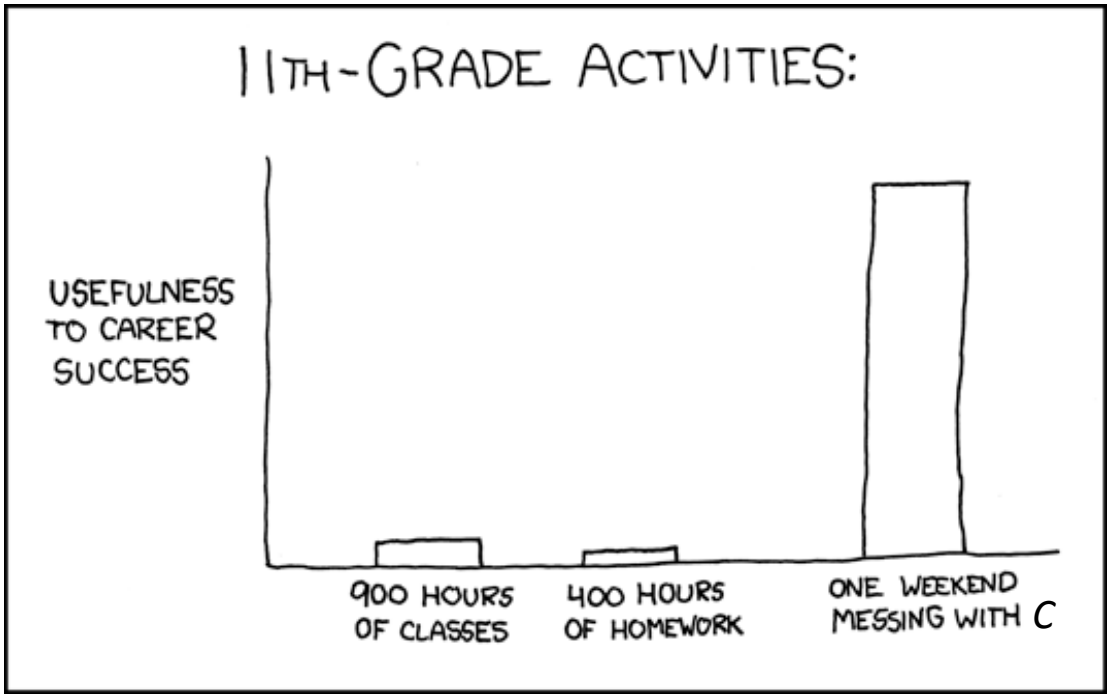
- Developed in 1972 by Dennis Ritchie at Bell Labs
- It is imperative programming language
- It provides:
  - Stability and small size code
  - Provide functionality through rich set of function libraries
  - Gateway for other professional languages like C++ and Java

Introduction to C

- It is used:
  - System software, Compilers, Editors, embedded systems, application programs
  - Data compression, graphics and computational geometry, utility programs
  - Databases, operating systems, device drivers, system level routines
- The real world still runs on C
  - Most of legacy code in use are in C
  - Many other programming languages are based on C

Introduction to C

- http://spectrum.ieee.org/computing/software/the-2015-top-ten-programming-languages



Original comic is available here: http://xkcd.com/519/

Basic C variable types

- There are five basic data types in C
  - Char: 'a'
    - A single byte capable of holding one character in the local character set
  - Int: 3
    - An integer of unspecified size
  - Float: 3.14
    - Single-precision floating point
  - Double: 3.1415926
    - Double-precision floating point
  - Void: Valueless special purpose type

Basic C variable types

Type (32 bit)	Smallest Value	Largest Value
short int	-32,768(-2^{15})	32,767(2^{15}-1)
unsigned short int	0	65,535(2^{16}-1)
Int	-2,147,483,648(-2^{31})	2,147,483,648(2^{31}-1)
unsigned int	0	4,294,967,295
long int	-2,147,483,648(-2^{31})	2,147,483,648(2^{31}-1)
unsigned long int	0	4,294,967,295

Variable assignment

- In C variables must be declared
- They are given values through assignments
- Assignment is done with the '=' operator

Declarations

```
int number_of_students;
float average_gpa;
```

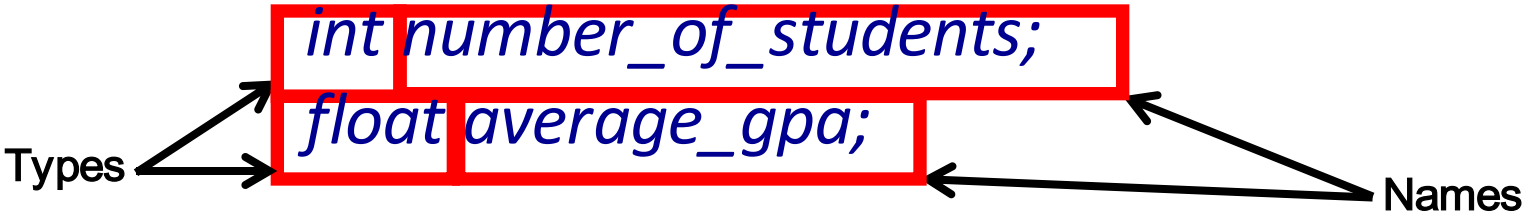
Assignments

```
number_of_students = 12;
average_gpa = 3.9;
```

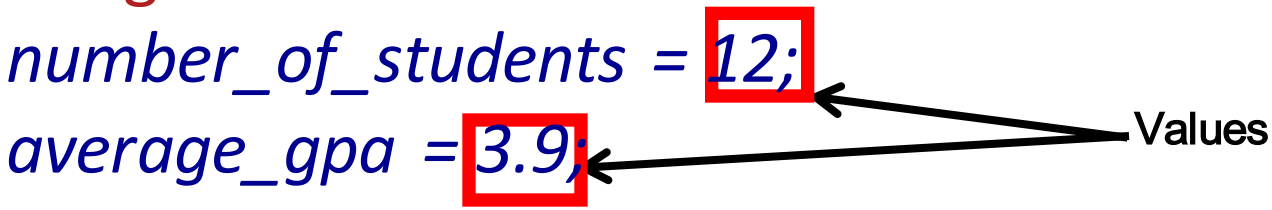
Variable assignment

- In C variables must be declared
- They are given values through assignments
- Assignment is done with the '=' operator

Declarations



Assignments



A Simple C Program

```
#include <stdio.h> /* Header files */

int main(void) {
    printf ("Hello World!\n" ) ;
    return 0;
}
```

C Program compilation

Compile: gcc – o myhello hello.c

Run: ./myhello

C Program Analysis

- #include <stdio.h> /* Header files */
  - It is a preprocessor directive
  - It tells computer to load contents of the file
  - It allows standard input/output operations
- Comments are used to describe program
  - Text surrounded by /* and */ is ignored by computer
  - Lines starting with // are also ignored

C Program Analysis

- int main (void)
  - C programs contain one or more functions, exactly one of which must be main
  - Parenthesis used to indicate a function
  - int means that main "returns" an integer value
- Braces ({ and }) indicate a block
  - Bodies of all functions must be contained in braces
- printf ("Hello World!\n")
  - printf and scanf functions

C Program Analysis

- prinf
  - Sends output to standard out
  - General form
    - printf(format descriptor, var1, var2, …);
  - printf("%s\n", "Hello world");
    - Translation: Print hello world as a string followed by a newline character
  - printf("%d\t%f\n", j, k);
    - Translation: Print the value of the variable j as an integer followed by a tab followed by the value of floating point variable k followed by a new line

C Program Analysis

- scanf
  - Gets inputs from user
  - General form
    - scanf(format descriptor, &var1, &var2, …);
  - scanf("%f", &i);
    - Translation: Get floating point input i from user
  - scanf("%d %f\n", &j, &k);
    - Translation: Get the value of the variable j as an integer followed by the value of floating point variable k from user
    - Blocks program until user enters input

C Program Analysis

- Some special characters are not visible directly in the output stream
- These begin with an escape character (\);
  - \n newline
  - \t horizontal tab
  - \a alert bell
  - \v vertical tab

C Program Operations

- Arithmetic operators
  - + "plus"
  - - "minus"
  - * "times"
  - / "divided by"

```
#include <stdio.h> /* Header files */
int number1, number2, number3;

int main(void) {
   scanf("Enter number1: %d", &number1);
   scanf("\nEnter number2: %d", &number2);
   number3 = number1 + number 2;
   printf ("\n Number1 + number 2 = %d\n", number3 ) ;

   number3 = number1 - number 2;
   printf ("\n Number1 - number 2 = %d\n", number3 ) ;

   number3 = number1 * number 2;
   printf ("\n Number1 * number 2 = %d\n", number3 ) ;

   number3 = number1 / number 2;
   printf ("\n Number1 / number 2 = %d\n", number3 ) ;
   return 0;
}
```

C Program Comparators

- Relational operators:
  - == "is equal to"
  - != "is not equal to"
  - > "greater than"
  - < "less than"
  - >= "greater than or equal to"
  - <= "less than or equal to"

C Program Logical Operators

- There are two logical operators in C
- || "logical or"
  - An expression formed with || evaluates to true if any one of its components is true
- && "logical and"
  - An expression formed with && evaluates to true if all of its components are true

Advance Data types

- In C
  - Arrays (a list of data (all of the Same Data Type!))
    - int grades [ ] = {94, 78, 88, 90, 93, 87, 59};
  - Structures (a collection of named data referring to a single entity)

```
struct Student {
  char Name [50] ;
  int id;
  float GPA;
  char major [25] ;
} ;
```

Advance Data types

- Pointers in C
  - Pointers are memory addresses
  - Every variable has a memory address
  - Symbol & means "take the address of" e.g., &x
  - Symbol * means "take the value of" e.g., *p
  - Symbol * is also used to denote a pointer type e.g., int *q;

Advance Data types

- Pointers in C
  - Declaration of integer pointers and and an integer number
    - int * pointer1 , * pointer2 ;
    - int number1;
  - Setting pointer1 equal to the address of number1
    - pointer1 = &number1;
  - Setting pointer2 equal to pointer1
    - pointer2 = pointer1;

Functions

- A Definition: A function is a named, independent section of C code that performs a specific task and optionally returns a value to the calling program or/and receives values(s) from the calling program
- There are two types of function
  - Predefined functions
    - Standard libraries like stdio.h, math.h
  - User-defined functions
    - Programmer created functions for specialized tasks
    - e.g., *int fibonacci(int n)*

Functions

- Characteristics of a function
- Function header: Its has a return type, a unique name, and list of parameters with their types

```
Return_type  function_name   (type1 parameter1, type2 parameter2
...){
        variable declaration(s)
        statement(s)
}
```

Examples

```
        void  function1 (int x, float y, char z)
        float  function2 (float x, double y)
        int     function3 (long size)
        void   function4 (void)
```

Functions

- The rules govern the use of variables in functions:
  - To use a variable in a function, it must be declared either in the function header or the function body
  - For a function to obtain a value from the calling program (caller), the value must be passed as an argument (the actual value) unless it is a global value

```
/* declare and define */
int exponential (int x)
{
  int result = 1;
  int i;
  for (i = 0, i< x, i++)
     result *= 2;
  return result;
}
int main()
{
  /* function call */
  int y;
  y = exponential(3);
}
```

Functions

- The rules govern the use of variables in functions:
  - For a calling program (caller) to obtain a value from function, the value must be explicitly returned from the called function (callee) unless it is updated through a global variable

```
/* declare and define */
int exponential (int x)
{
  int result = 1;
  int i;
  for (i = 0, i< x, i++)
     result *= 2;
  return result;
}
int main()
{
  /* function call */
  int y;
  y = exponential(3);
}
```

Recursion

- Often it is difficult to express a problem explicitly
  - For example the Fibonacci sequence: 0,1,1,2,3,5,8,13,21,34,55,…
  - It is difficult to follow the logic of this sequence
- However, a recursive definition consisting of expressing higher terms in the sequence in terms of lower terms
  - Recursive definition for $\{f_n\}$:
  - Initialization: $f_0 = 0, f_1 = 1$
  - Recursion: $n = f_{n-1} + f_{n-2}$ for $n > 1$

Recursion

- Sometimes the best way to solve a problem is by solving a smaller version of the exact same problem first
- Recursion is a technique that solves a problem by solving a smaller problem of the same type
- The technique ends up with functions that call themselves (recursive functions)

Logic of recursive functions

- Recursive definition and inductive proofs are complement each other
- A recursive function has two parts
- Initialization – analogous to induction base cases
- Recursion – analogous to induction step
  - Recursive definition for $\{f_n\}$:
  - Initialization: $f_0 = 0, f_1 = 1$
  - Recursion: $n = f_{n-1} + f_{n-2}$ for $n > 1$

Recursion

- Factorial function
  - Iterative implementation

```
int Factorial(int n)
{
   int count;
   int fact = 1;
   for(count = 2; count <= n; count++)
        fact = fact * count;
   return fact;
}
```

Recursion

- Factorial function
  - Recursive implementation

```
int Factorial(int n)
{
   if (n==0)  // base case
      return 1;
   else
      return n * Factorial(n-1);
}
```

What does any language need to do?

Language Perspective

1. Declare and initialize variables
2. Access variables
3. Control flow of execution
4. Use data structures
5. Execute statements

Potential Attack Vectors

Next Class

- Programming & Computer Organization