Computer Systems DV1 (1DT151)
Operating Systems (1DT020)

Cary Laxer, Ph.D.

Visiting Lecturer

Today’s class

- Introductions
- Review of some C
- Computer system overview

Introductions

Instructor

- Cary Laxer
- Visiting lecturer
- Home institution is Rose-Hulman Institute of Technology, Terre Haute, Indiana, USA
- Professor and Head of Computer Science and Software Engineering
- Bachelor’s degree in computer science and mathematics from New York University
- Ph.D. in biomedical engineering from Duke University

UPPSALA
UNIVERSITET

Informationsteknologi

Lab instructor

- John Håkansson
- Ph.D. student in the department
- M.Sc. in 2000
- Industry experience writing C compilers for embedded systems and as a robot programmer
- Has assisted teaching this course before

Course

- Information is maintained on the course website: www.it.uu.se/edu/course/homepage/datsystDV/ht07
- 12 lecture meetings and 4 lab meetings
- Text is *Operating Systems: Internals and Design Principles (Fifth Edition)* by William Stallings
- We will cover chapters 1-10, 12, and 16
- I will try to have some in-class exercises to help reinforce the material and to break up the long lecture periods

UPPSALA UNIVERSITET

Informationsteknologi

Introduce yourselves

- Tell us:
  - Your name
  - Your hometown
  - Your computer background
  - Something interesting about yourself

Review of C

Why learn C?

- The good…
  - Both a high-level and a low-level language
  - Better control of low-level mechanisms
  - Performance better than Java
  - Java hides many details needed for writing OS code
- And the bad…
  - Memory management responsibility is yours
  - Explicit initialization and error detection
  - More room for mistakes

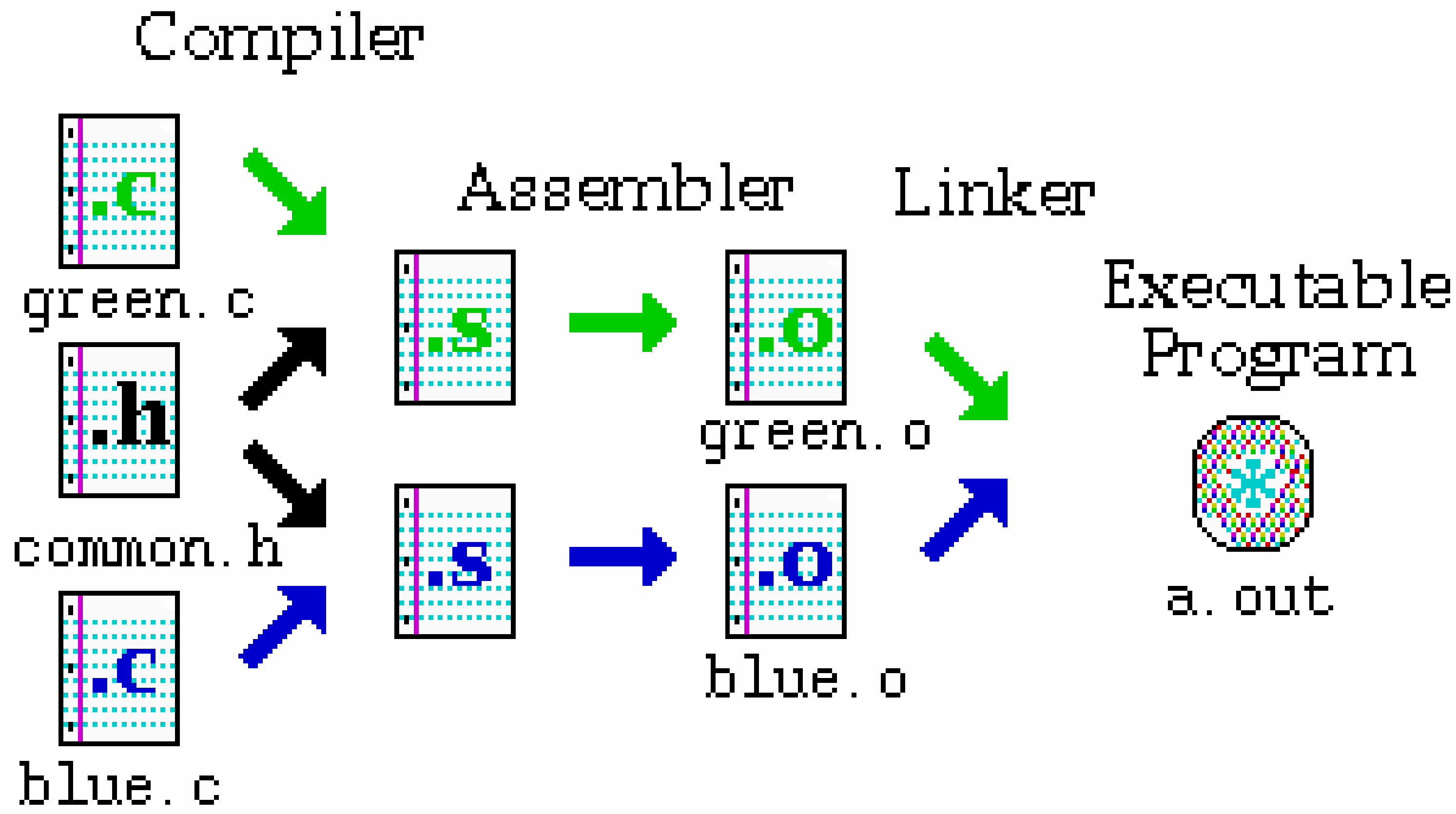
Goals of this review

- To review (introduce if you are new to C) some basic C concepts to you
  - so that you can read further details on your own
- To warn you about common mistakes made by beginners

UPPSALA UNIVERSITET

Informationsteknologi

Creating an executable



Source: http://www.eng.hawaii.edu/Tutor/Make/1-2.html

Types of files

- C source files (.c)
- C header files (.h)
- Object files (.o)
- Executable files (typically no extension – by default : a.out)
- Library files (.a or .so)

Example 1

```
#include <stdio.h>      //#include “myheader.h”

int
main()
{
    printf(“Hello World. \n \t and you ! \n ”);
              /* print out a message */
    return 0;
}
```

Summarizing the Example

- #include <stdio.h> = include header file stdio.h
  - No semicolon at end
  - Small letters only – C is case-sensitive
- int main(){ … } is the only code executed
- printf(“ /* message you want printed */ ”);
- \n = newline \t = tab
- \ in front of other special characters within printf creates “escape sequences”.
  - printf(“Have you heard of \”The Rock\” ? \n”);

Compiling and running

- >gcc ex1.c (Creates a.out)
- >./a.out (Runs the executable)

- >gcc ex1.c –o ex1 (Creates ex1 not a.out)
- >./ex1

External library files libname.a or libname.so

- Special functionality is provided in the form of external libraries of ready-made functions
- Ready-compiled code that the compiler merges, or links, with a C program during compilation
- For example, libraries of mathematical functions, string handling functions, and input/output functions
- Look for the library files under /usr/lib and header files under /usr/include

External library files libname.a or libname.so

- To compile, use flag “l” and name i.e. –lname.

  eg. gcc –o test test.c –lm

  where “m” in “lm” comes from libm.so i.e. the math library.
- .a libraries are static – code is included in the executable program
- .so libraries are dynamic – code is not in the executable program; the system copy is used at run time

UPPSALA UNIVERSITET

Informationsteknologi

Using external library files

- To use the library files, you must always do two things:
  - link the library with a -l option to gcc
  - include the library header files

Pre-processor directives

- A preprocessor is a program that examines C code before it is compiled and manipulates it in various ways.
- Two main functions
  - To include external files using #include
  - To define macros (names that are expanded by the preprocessor into pieces of text or C code) using #define

UPPSALA UNIVERSITET

Informationsteknologi

Example of pre-processor directives

Example 2:

```
#include <stdio.h>
#define STRING1 "A macro definition\n"
#define STRING2 "must be all on one line!\n"
#define EXPRESSION1 1 + 2 + 3 + 4
#define EXPRESSION2 EXPRESSION1 + 10
#define ABS(x) ((x) < 0) ? -(x) : (x)
#define MAX(a,b) (a < b) ? (b) : (a)
#define BIGGEST(a,b,c) (MAX(a,b) < c) ? (c) : (MAX(a,b))

int
main ()
{
 printf (STRING1);
 printf (STRING2);
 printf ("%d\n", EXPRESSION1);
 printf ("%d\n", EXPRESSION2);
 printf ("%d\n", ABS(-5));
 printf ("Biggest of 1, 2, and 3 is %d\n", BIGGEST(1,2,3));
 return 0;
}
```

#define

- The expression is NOT evaluated when it replaces the macro in the pre-processing stage.
- Evaluation takes place only during the execution phase.

Simple Data Types

Data Type	# bytes (typical)	Shorthand
int	4	%d %i
char	1	%c
float	4	%f
double	8	%lf
long	4	%l
short	2	%i

String - %s
address - %p(HEX) or %u (unsigned int)

Example 3

```
#include <stdio.h>

int
main()
{
        int nstudents = 0; /* Initialization, required */
        float age = 21.527;

        printf(“How many students does Uppsala University have ?”);
        scanf (“%d”, &nstudents);  /* Read input */
        printf(“Uppsala University has %d students.\n”, nstudents);
        printf(“The average age of the students is %3.1f\n”,age);
                                //3.1 => width.precision
        return 0;
}
```

```
>./ex3
How many students does Uppsala University have ?:2000 (enter)
Uppsala University has 2000 students.
The average age of the students is 21.5
>
```

If you are familiar with Java...

- Operators same as Java:
  - Arithmetic
    - `int i = i+1; i++; i--; i *= 2;`
    - `+, -, *, /, %`
  - Relational and Logical
    - `<, >, <=, >=, ==, !=`
    - `&&, ||, &, |, !`
- Syntax same as in Java:
  - `if ( ) { } else { }`
  - `while ( ) { }`
  - `do { } while ( );`
  - `for (i=1; i <= 100; i++) { }`
  - `switch ( ) {case 1: … }`
  - `continue; break;`

Example 4

```
#include <stdio.h>
#define DANGERLEVEL 5   /* C Preprocessor -
                           - substitution on appearance */
int
main()
{
    float level=1;
    if (level <= DANGERLEVEL){ /*replaced by 5*/
        printf(“Low on gas!\n”);
        }
        else printf(“On my way !\n”);

    return 0;
}
```

One-Dimensional Arrays

Example 5:

```
#include <stdio.h>

int
main()
{
   int number[12]; /* 12  numbers*/
   int index, sum = 0;
            /* Always initialize array before use */
   for (index = 0; index < 12; index++) {
      number[index] = index;
   }
   /* now, number[index]=index; will cause error:why ?*/

   for (index = 0; index < 12; index = index + 1) {
      sum += number[index];  /* sum array elements */
   }

   return 0;
}
```

More arrays - Strings

- char name[10]; //declaration
- name = {‘A’,’l’,’i’,’c’,’e’,’\0’}; //initialization
  /* ’\0’= end of string */
- char name [] = “Alice”; //declaration and initialization
- char name [] = {‘A’,’l’,’i’,’c’,’e’,’\0’}; // ditto
- scanf(“%s”,name); //Initialization
  // ERROR: scanf(“%s”,&name);
- printf(“%s”, name); /* print until ‘\0’ *

Strings continued

- Functions to operate on strings
  - strcpy, strncpy, strcmp, strncmp, strcat, strncat, substr, strlen,strtok
  - #include <strings.h> or <string.h> at program start
- CAUTION: C allows strings of any length to be stored. Characters beyond the end of the array will overwrite data in memory following the array.

Multi-dimensional arrays

- int points[3][4];
- points [1][3] = 12; /* NOT points[3,4] */
- printf(“%d”, points[1][3]);

Computer system overview

UPPSALA UNIVERSITET

Informationsteknologi

Operating System

- Exploits the hardware resources of one or more processors
- Provides a set of services to system users
- Manages secondary memory and I/O devices

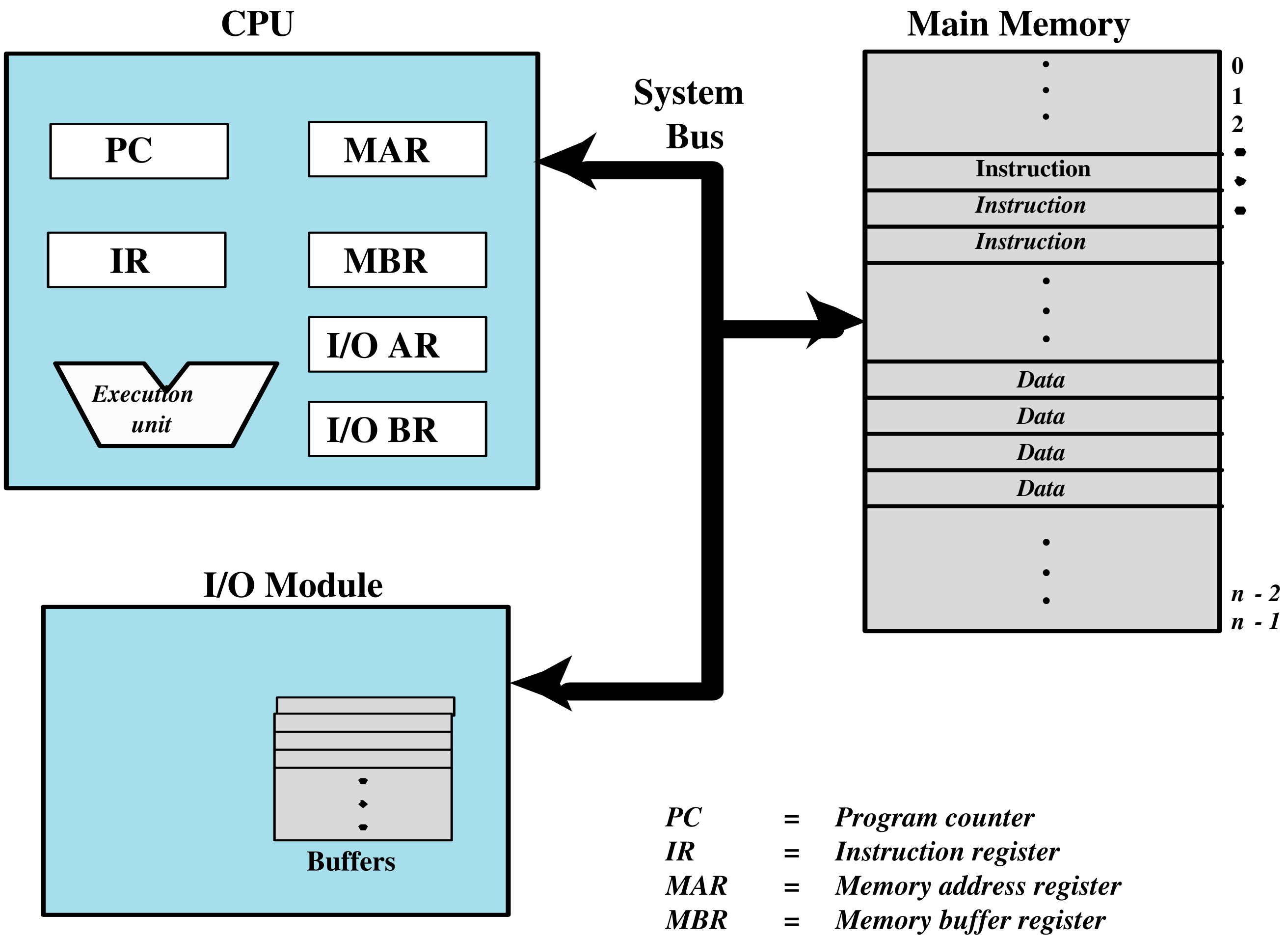
Basic Elements

- Processor
- Main Memory
  - volatile
  - referred to as real memory or primary memory
- I/O modules
  - secondary memory devices
  - communications equipment
  - terminals
- System bus
  - communication among processors, memory, and I/O modules

Processor

- Two internal registers
  - Memory address register (MAR)
    - Specifies the address for the next read or write
  - Memory buffer register (MBR)
    - Contains data written into memory or receives data read from memory
  - I/O address register
  - I/O buffer register

Top-Level Components



PC	=	*Program counter*
IR	=	*Instruction register*
MAR	=	*Memory address register*
MBR	=	*Memory buffer register*
I/O AR	=	*Input/output address register*
I/O BR	=	*Input/output buffer register*

Figure 1.1 Computer Components: Top-Level View

Processor Registers

- User-visible registers
  - Enable programmer to minimize main-memory references by optimizing register use
- Control and status registers
  - Used by processor to control operating of the processor
  - Used by privileged operating-system routines to control the execution of programs

User-Visible Registers

- May be referenced by machine language
- Available to all programs - application programs and system programs
- Types of registers
  - Data
  - Address
    - Index
    - Segment pointer
    - Stack pointer

User-Visible Registers

- Address Registers
  - Index
    - Involves adding an index to a base value to get an address
  - Segment pointer
    - When memory is divided into segments, memory is referenced by a segment and an offset
  - Stack pointer
    - Points to top of stack

UPPSALA UNIVERSITET

Informationsteknologi

Control and Status Registers

- Program Counter (PC)
  - Contains the address of an instruction to be fetched
- Instruction Register (IR)
  - Contains the instruction most recently fetched
- Program Status Word (PSW)
  - Condition codes
  - Interrupt enable/disable
  - Supervisor/user mode

Control and Status Registers

- Condition Codes or Flags
  - Bits set by the processor hardware as a result of operations
  - Examples
    - Positive result
    - Negative result
    - Zero
    - Overflow