CS354: Machine Organization and Programming

Lecture 2
Friday the September 4th 2015

Section 2
Instructor: Leo Arulraj

Class Announcements

1. Are your mailing lists working ? Did you receive the Welcome email yesterday evening ?
2. Question about Midterm 1 conflict: How many are taking ECE 353?
3. Code, slides, etc. will be shared with you but it won’t be timely. So, take notes in class!
4. Good job in Piazza (Anon. posts are okay!)
5. **Project 0 is due Sep 14th before 9 AM.** Partner details for projects.

Five Realities you must embrace

1. Ints are not Ints and Floats are not reals.
2. You have to know assembly.
3. Memory hierarchy matters
4. Performance is not just algorithmic complexity
5. Computer do more than just load-store-execute! They do I/O, networking with other computers etc. for example.

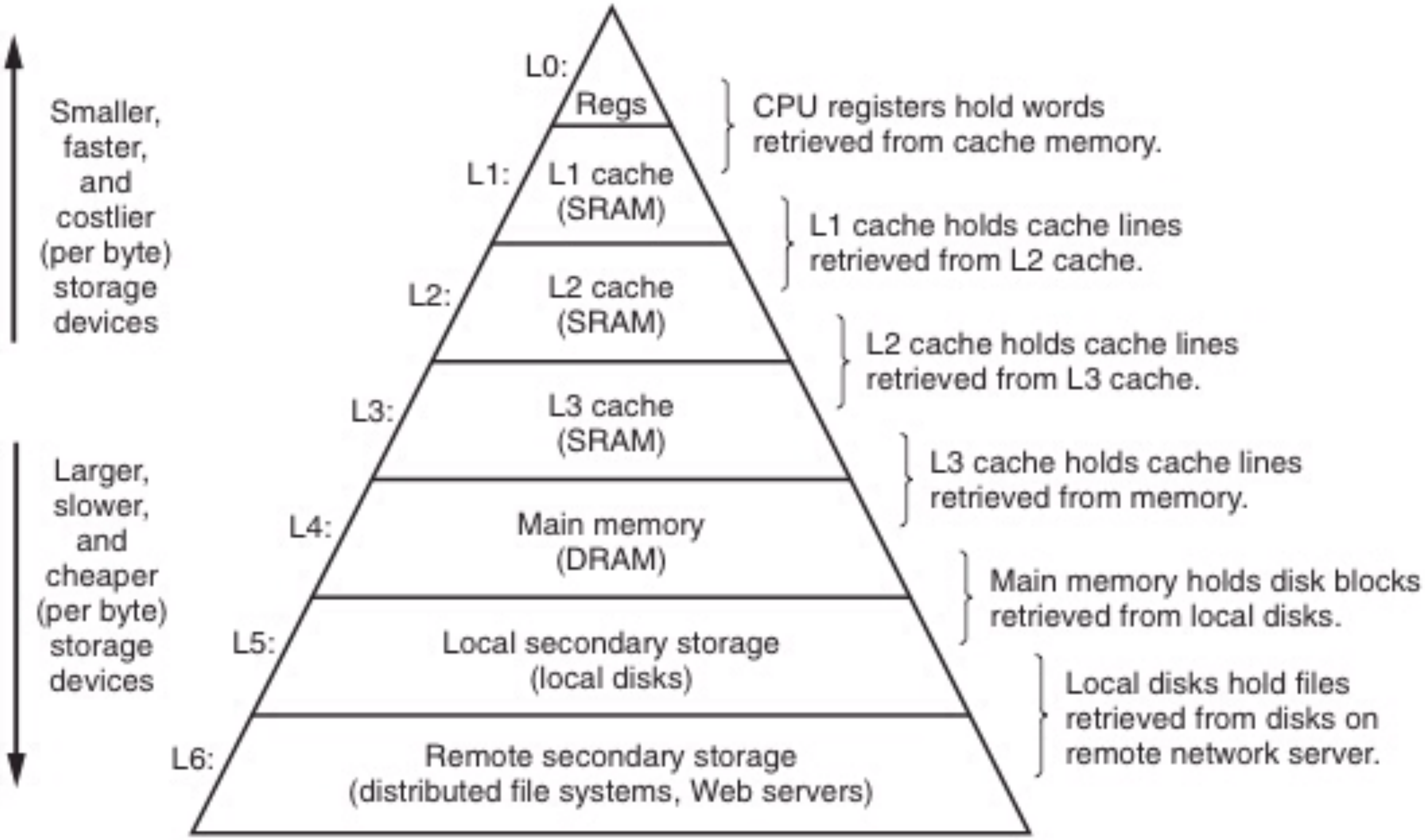
Reality 3

Great Reality #3: Memory Matters

Random Access Memory Is an Unphysical Abstraction

- Memory is not unbounded
  - It must be allocated and managed
  - Many applications are memory dominated
- Memory referencing bugs especially pernicious
  - Effects are distant in both time and space
- Memory performance is not uniform
  - Cache and virtual memory effects can greatly affect program performance
  - Adapting program to characteristics of memory system can lead to major speed improvements

Memory Hierarchy


Smaller, faster, and costlier (per byte) storage devices
Larger, slower, and cheaper (per byte) storage devices
L0:
Regs
L1:
L1 cache (SRAM)
L2:
L2 cache (SRAM)
L3:
L3 cache (SRAM)
L4:
Main memory (DRAM)
L5:
Local secondary storage (local disks)
L6:
Remote secondary storage (distributed file systems, Web servers)

CPU registers hold words retrieved from cache memory.

L1 cache holds cache lines retrieved from L2 cache.

L2 cache holds cache lines retrieved from L3 cache.

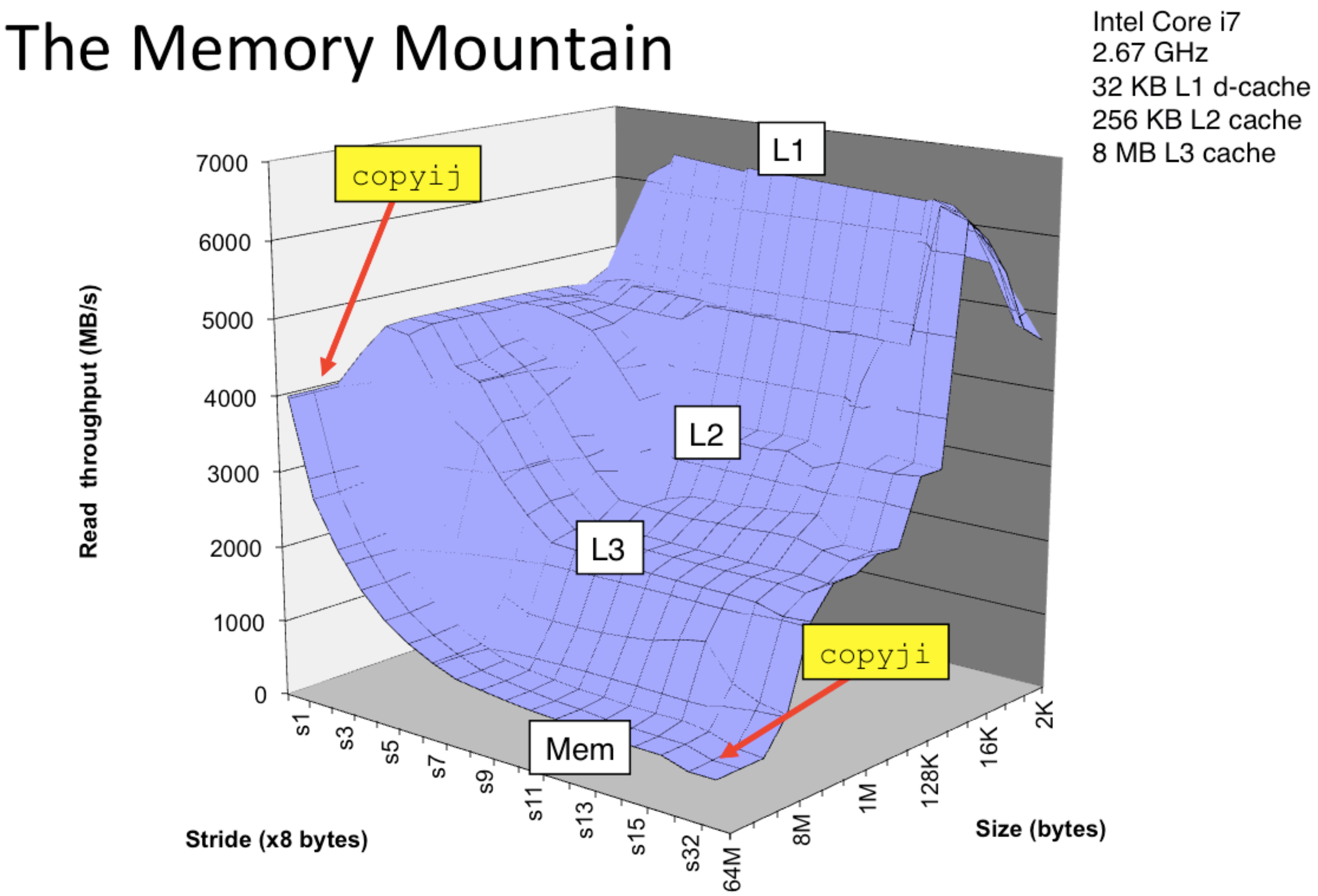
L3 cache holds cache lines retrieved from memory.

Main memory holds disk blocks retrieved from local disks.

Local disks hold files retrieved from disks on remote network server.

Figure 1.9 **An example of a memory hierarchy.**

Memory Mountain



The Memory Mountain

Intel Core i7
2.67 GHz
32 KB L1 d-cache
256 KB L2 cache
8 MB L3 cache

Introduction to C Programming

- Operators
- Types and Declarations
- Statements
- Functions
- C-Preprocessor Directives
- Simple I/O

Operators

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Bitwise Operators
- Assignment Operators
- Miscellaneous Operators

Arithmetic Operators

Op.	Description	Example A=10,B=20
+	Adds two operands	A + B will give 30
-	Subtracts second operand from the first	A - B will give -10
*	Multiplies both operands	A * B will give 200
/	Divides numerator by de-numerator	B / A will give 2
%	Modulus Operator and remainder of after an integer division	B % A will give 0
++	Increments operator increases integer value by one	A++ will give 11
--	Decrements operator decreases integer value by one	A-- will give 9

Relational Operators

Op.	Description	Example A=10, B=20
==	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

Logical Operators

Op.	Description	Example A=true, B=false
&&	Called Logical AND operator. If both the operands are non-zero, then condition becomes true.	(A && B) is false.
||	Called Logical OR Operator. If any of the two operands is non-zero, then condition becomes true	(A || B) is true.
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B) is true.

Bitwise Operators

Op.	Description	Example A(60) = 0011 1100 B(13) = 0000 1101
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) will give 12, which is 0000 1100
|	Binary OR Operator copies a bit if it exists in either operand.	(A | B) will give 61, which is 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) will give 49, which is 0011 0001
~	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) will give -61, which is 1100 0011 in 2's complement form.
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 will give 240 which is 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 will give 15 which is 0000 1111

Assignment Operators 1

Op.	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	C = A + B will assign value of A + B into C
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand	C /= A is equivalent to C = C / A

Assignment Operators 2

Op.	Description	Example
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator	C &= 2 is same as C = C & 2
^=	bitwise exclusive OR and assignment operator	C ^= 2 is same as C = C ^ 2
|=	bitwise inclusive OR and assignment operator	C |= 2 is same as C = C | 2

Miscellaneous Operators

Op.	Description	Example
sizeof()	Returns the size of an variable.	sizeof(a), where a is integer, will return 4.
Unary &	Returns the address of an variable.	&a; will give actual address of the variable.
Unary *	Value of a pointer	*a; will value stored in the address a.
? :	Conditional Expression	If Condition is true ? Then value X : Otherwise value Y

Example C program on Operators

Operator Precedence

a>b+c&&d

This expression is equivalent to:

((a>(b+c))&&d)

Why not this?: ((a>b)+(c&&d))

Operator Precedence 1

Operator Name	Associativity	Operators
Primary scope resolution	left to right	::
Primary	left to right	() [ ] . -> dynamic_cast typeid
Unary	right to left	++ -- + - ! ~ & * (type_name) sizeof new delete
C++ Pointer to Member	left to right	.*->*
Multiplicative	left to right	* / %
Additive	left to right	+ -
Bitwise Shift	left to right	<< >>
Relational	left to right	< > <= >=

Operator Precedence 2

Operator Name	Associativity	Operators
Equality	left to right	== !=
Bitwise AND	left to right	&
Bitwise Exclusive OR	left to right	^
Bitwise Inclusive OR	left to right	|
Logical AND	left to right	&&
Logical OR	left to right	||
Conditional	right to left	? :
Assignment	right to left	= += -= *= /= <<= >>= %= &= ^= |=
Comma	left to right	,

Example C program on Operator precedence

Types

- Integer types
- Floating point types
- The void type
- Type Qualifiers
- Strings in C

Integer Types

The actual size of integer types varies by implementation. Standard only requires size relations between the data types and minimum sizes for each.

Type	Storage size	Value range
char	1 byte	-128 to 127 or 0 to 255
unsigned char	1 byte	0 to 255
signed char	1 byte	-128 to 127
int	2 or 4 bytes	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
unsigned int	2 or 4 bytes	0 to 65,535 or 0 to 4,294,967,295
short	2 bytes	-32,768 to 32,767
unsigned short	2 bytes	0 to 65,535
long	4 bytes	-2,147,483,648 to 2,147,483,647
unsigned long	4 bytes	0 to 4,294,967,295

Floating Point Types

The value representation of floating-point types is implementation-defined

Type	Storage size	Value range	Precision
float	4 byte	1.2E-38 to 3.4E +38	6 decimal places
double	8 byte	2.3E-308 to 1.7E +308	15 decimal places
long double	10 byte	3.4E-4932 to 1.1E +4932	19 decimal places

Void type

1	**Function returns as void**
	There are various functions in C which do not return value or you can say they return void. A function with no return value has the return type as void. For example, **void exit (int status);**
2	**Function arguments as void**
	There are various functions in C which do not accept any parameter. A function with no parameter can accept as a void. For example, **int rand(void);**
3	**Pointers to void**
	A pointer of type void * represents the address of an object, but not its type. For example a memory allocation function **void *malloc(size_t size);** returns a pointer to void which can be casted to any data type.

Strings in C

- Strings in C are one dimensional arrays of characters terminated with a null character.

```
  Examples: char greeting[6] = {'H', 'e', 'l',
'l', 'o', '\0'};

            char greeting[6] = “Hello”;

            char* greeting = “Hello”;
```

Index	**0**	**1**	**2**	**3**	**4**	**5**
Content	H	e	l	l	o	\0
Memory Address.	0x88321	0x88322	0x88323	0x88324	0x88325	0x88326

Type Qualifiers

- const: means that something is not modifiable, so a data object that is declared with **const** as a part of its type specification must not be assigned to in any way during the run of a program.
- volatile: tells the compiler that the object is subject to sudden change for reasons which cannot be predicted from a study of the program itself, and forces every reference to such an object to be a genuine reference.
- restrict: Has to do with pointers. Later !

Storage Classes

#	Storage Specifier	Storage place	Initial / default value	Scope	Life
1	**auto**	memory	Garbage value	local	Within the function only.
2	**extern**	memory	Zero	Global	Till the end of the main program. Variable definition might be anywhere in the C program
3	**static**	memory	Zero	local	Retains the value of the variable between different function calls.
4	**register**	Register memory	Garbage value	local	Within the function

Declarations

Global Variable: A global variable is a variable that is declared outside **all** functions.

Local Variable: A local variable is a variable that is declared inside a function.

Examples:

```
const int foo = 10;
// foo is const integer with value 10

char foo;
// foo is a char

double foo();
// foo is a function returning a double
```

Explicit Type Conversions

```
double da = 3.3;

double db = 3.3;

double dc = 3.4;

int r1 = (int)da + (int)db + (int)dc; //r1 == 9

int r2 = (da + db + dc); // r2 == 10
```

Example C program on Types, sizes

If Statement

```
if(boolean_expression){
        /* statement(s) will execute if the
           boolean expression is true */
}
```

If-else Statement

```
if(boolean_expression){
        /* statement(s) will execute if the boolean
           expression is true */
}else{
        /* statement(s) will execute if the boolean
           expression is false */
}
```

Else-if Statement

```
if(expression){
    /*Block of statements;*/
}else if(expression){
    /*Block of statements;*/
}else{
    /*Block of statements;*/
}
```

Example C program on if, if else, else if statements

Switch

```
switch(expression){
      case constant-expression1:
statements1;        [case constant-
expression2:    statements2;]           [case
constant-expression3:    statements3;]
      [default: statements4;]
}
```

While loop

```
while (expression) {
    Single statement or
    Block of statements;
}
```

For loop

```
for(expression1;expression2;expression3){
    Single statement   or
    Block of statements;
}
```

You can also skip expression1, expression2, expression3.

What does this do ? **for(;;){printf("a\n");}**

Do while loop

```
do{
    Single statement    or
    Block of statements;
}while(expression);
```

Break; Continue; Statements

C provides two commands to control how we loop:

- **break --** exit form loop or switch.
- **continue --** skip 1 iteration of loop.

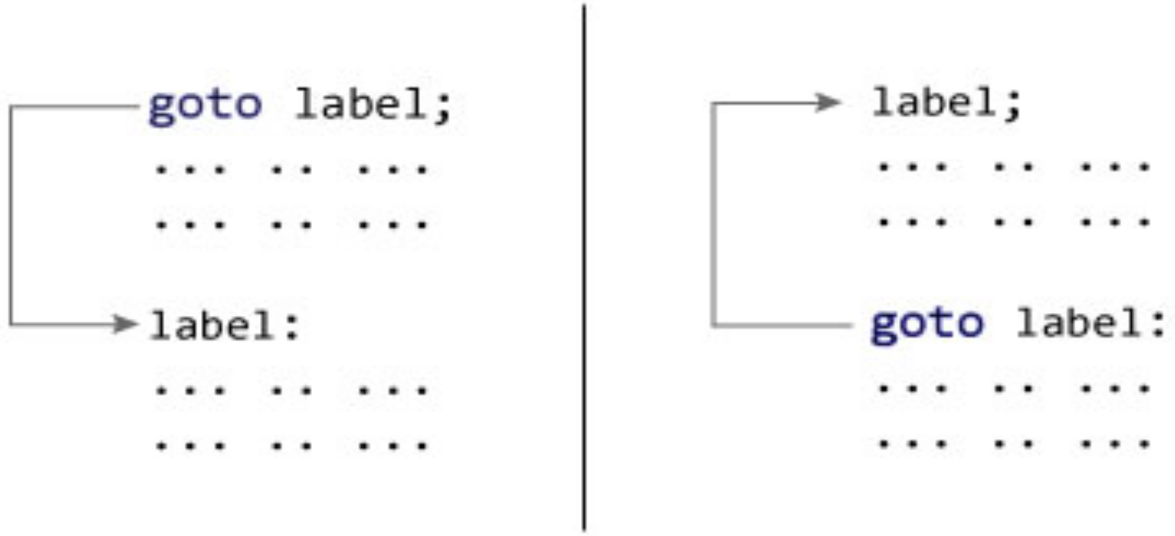
Goto and Labels

goto label;

....

....

....
label: statement



```
goto label;
... .. ...
... .. ...

label:
... .. ...
... .. ...
```

```
label;
... .. ...
... .. ...

goto label:
... .. ...
... .. ...
```

You can have better label names
(e.g. mycalc, complexcalc etc.)

Example C program on for, while loops, switch statements

Functions 1

Function Prototype (Declaration):

```
return_type function_name(
        type(1) argument(1),….,type(n) argument(n));
```

Function Definition:

```
return_type function_name(
type(1) argument(1),..,type(n) argument(n))
{
//body of function
}
```

Functions 2

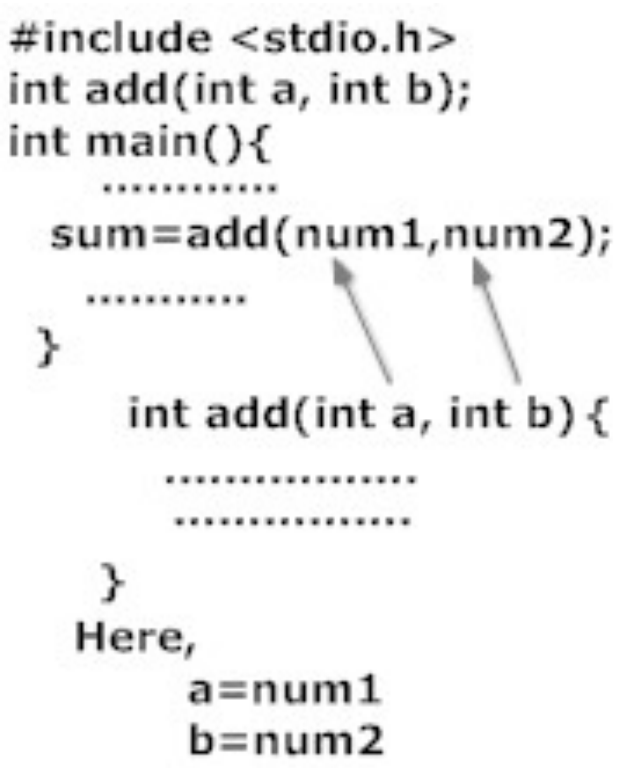
Function Call:
function_name(argument(1),....argument(n));

Return Statement:
return (expression);

C always passes arguments `by value': a copy of the value of each argument is passed to the function; the function cannot modify the actual argument passed to it.

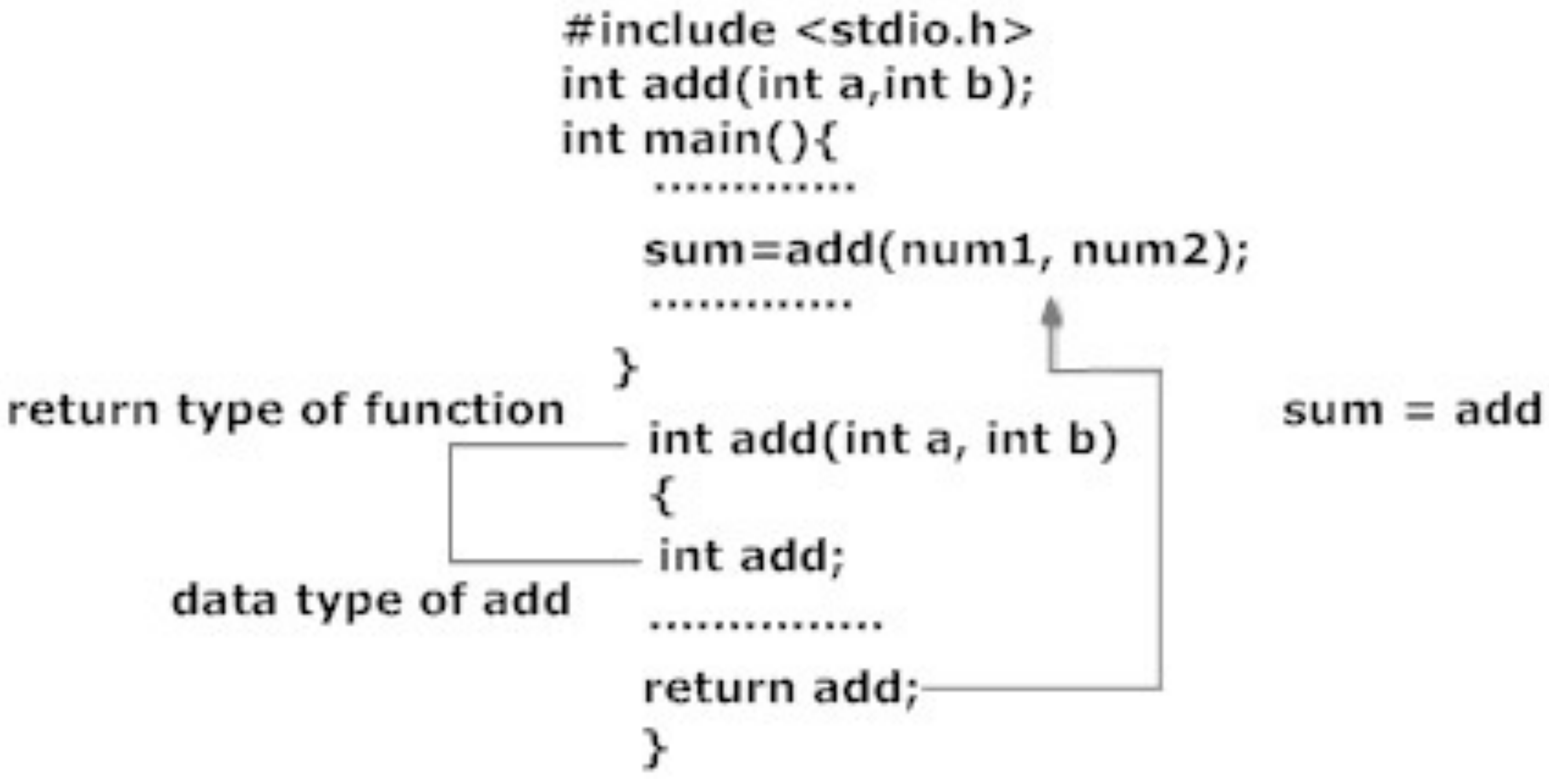
Functions

C always passes arguments `by value': a copy of the value of each argument is passed to the function; the function cannot modify the actual argument passed to it.



```
#include <stdio.h>
int add(int a, int b);
int main(){
      ............
   sum=add(num1,num2);
     ...........
 }
        int add(int a, int b){
          ................
          ...............
      }
    Here,
           a=num1
           b=num2
```

Working of Functions



```
#include <stdio.h>
int add(int a,int b);
int main(){
     .............
     sum=add(num1, num2);
     .............
   }
     int add(int a, int b)
     {
      int add;
     ..............
     return add;
     }
```

C Preprocessor

File Inclusion

#include <file> - used for system header files. File is looked for in standard list of system directories

#include “file” - used for local header files in program.

C Preprocessor

Macro substitution

#define [identifier name] [value]

Eg. #define PI_PLUS_ONE   (3.14 + 1)

#define MACRO_NAME(arg1, arg2, ...) [code to expand to]

Eg. #define MULT(x, y)   x * y

C Preprocessor

Conditional Inclusion:

Simple example is:

```
#ifdef MACRO
        controlled text
#endif /* MACRO */
```

More versions with else, ifndef etc. allowed.

Example C program illustrating C Preprocessor

Simple I/O

int printf(const char *format, …) function writes output to the standard output stream **stdout** and produces output according to a format provided.

int scanf(const char *format, …) function reads input from the standard input stream **stdin** and scans that input according to **format** provided.

Simple I/O Example

```
int b, a; long int b; char s[10], float d;

printf("%d\n",b);

scanf("%d", &a);

printf("%3d\n",b);

printf("%3.2f\n",d);

printf(”%ld\n",b);
```

I/O Redirection and Pipes

I/O Redirection:

prog <infile >outfile
infile will be stdin and outfile will be stdout

Pipes: |

With pipes, the standard output of one command is fed into the standard input of another.

Format String 1

Specifier	Description	Example
%i or %d	int	12345
%c	char	y
%s	string	“sdfa”
%f	Display the floating point number using decimal representation	3.1415
%e	Display the floating point number using scientific notation with e	1.86e6
%E	Like e, but with a capital E in the output	1.86E+06
%g	Use shorter of the 2 representations: f or e	3.1 or 1.86e6
%G	Like g, except uses the shorter of f or E	3.1 or 1.86E6

Format String 2

Variable type	Length Modifier	Example

short int, unsigned short int	h	short int i = 3;
		printf("%hd", i);

long int or unsigned long int	l	long int i = 3;
		printf("%ld", i);

wide characters or strings	l	wchar_t* wide_str = L"Wide String";
		printf("%ls", wide_str);

long double	L	long double d = 3.1415926535;
		printf("%Lg", d);

Example C program with simple I/O

Format Specifiers have a ton more details

Eg. http://en.cppreference.com/w/cpp/io/c/fprintf

Comments in C

- Single Line Comments:

```
// this is a single line comment
```

- Multi Line Comments:

```
/* this is a multi
        line
        comment */
```

Undefined Behavior

The C FAQ defines “undefined behavior” like this:

Anything at all can happen; the Standard imposes no requirements. The program may fail to compile, or it may execute incorrectly (either crashing or silently generating incorrect results), or it may fortuitously do exactly what the programmer intended.

Undefined Behavior Example

As a quick example let’s take this program:

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

int main (void)
{
  printf ("%d\n", (2147483647+1) < 0);
  return 0;
}
```

Undefined Behavior
Allowed Results

```
$ ./test
1

$ ./test
0

$ ./test
42
```

And this:

```
$ ./test
Formatting root partition, chomp chomp
```

See you in Next Lecture

- Read Chapter 1 in K&R (C Programming Language Book)
- Read more of K&R (Ch 2-7)
- Try out some examples on your own, understand what they do line by line
- **Start early on Assignment 0!**