CS 103 Lecture 2 Slides

C/C++ Basics

Mark Redekopp

Announcements

- Get your VM's installed.
  - Do's and Don'ts with your VM
    - Installing the 'Guest Additions' for the Linux VM
    - Backing up files
    - Not installing any updates to the VM
- HW 1
- Lab 1 review answers must be submitted on our website
  - Attend lab to meet your TAs and mentors and get help with lab 1 or your VM

A quick high-level view before we dive into the details…

PROGRAM STRUCTURE AND COMPILATION PROCESS

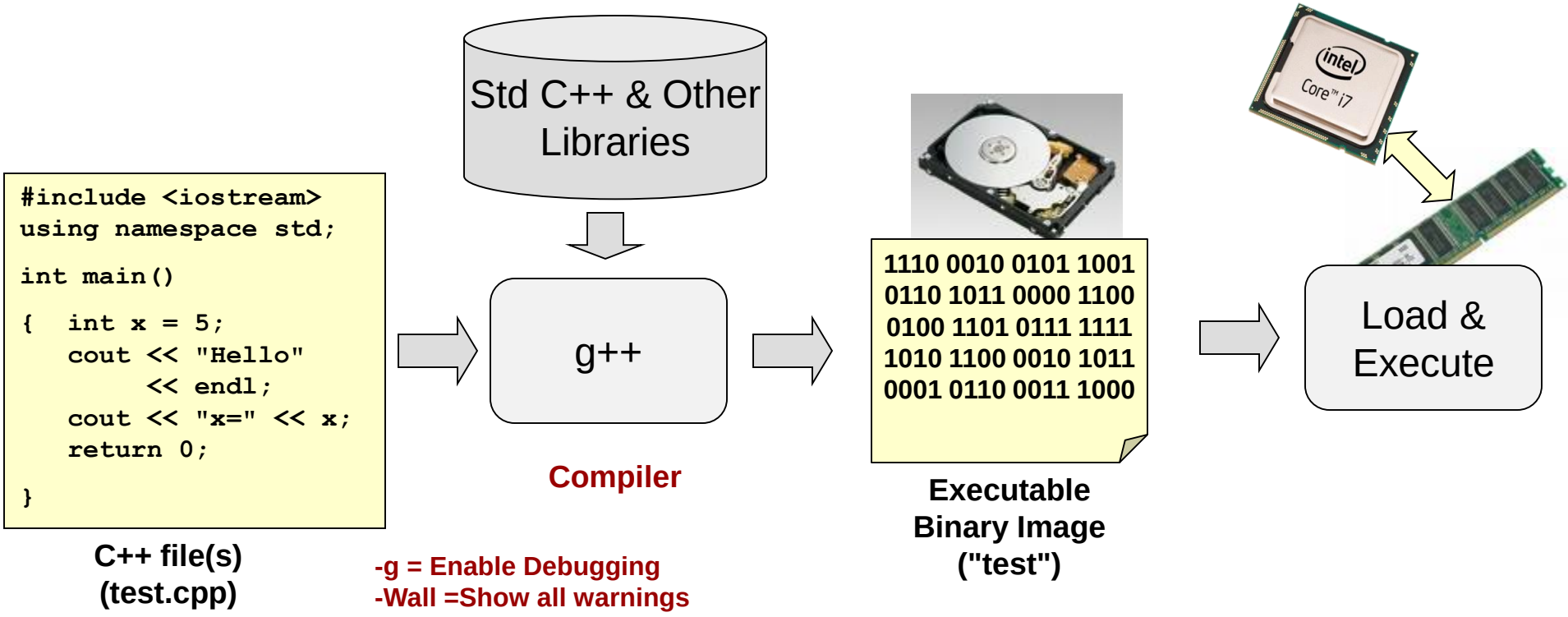
C/C++ Program Format/Structure

- Comments
  - Anywhere in the code
  - C-Style => "/*" and "*/"
  - C++ Style => "//"
- Compiler Directives
  - #includes tell compiler what other library functions you plan on using
  - 'using namespace std;' -- Just do it for now!
- main() function
  - Starting point of execution for the program
  - All code/statements in C must be inside a function
  - Statements execute one after the next and end with a semicolon (;)
  - Ends with a 'return 0;' statement
- Other functions
  - printName() is a function that can be "called"/"invoked" from main or any other function

```
/* Anything between slash-star and
star-slash is ignored even across
multiple lines of text or code */

// Anything after "//" is ignored on a line

// #includes allow access to library functions
#include <iostream>
#include <cmath>
using namespace std;

void printName()
{
  cout << "Tommy Trojan" << endl;
}

// Execution always starts at the main() function
int main()
{
  cout << "Hello: " << endl;
  printName();
  printName();
  return 0;
}
```

```
Hello:
Tommy Trojan
Tommy Trojan
```

Software Process




```
$ gedit test.cpp &
```

1 **Edit & write code**

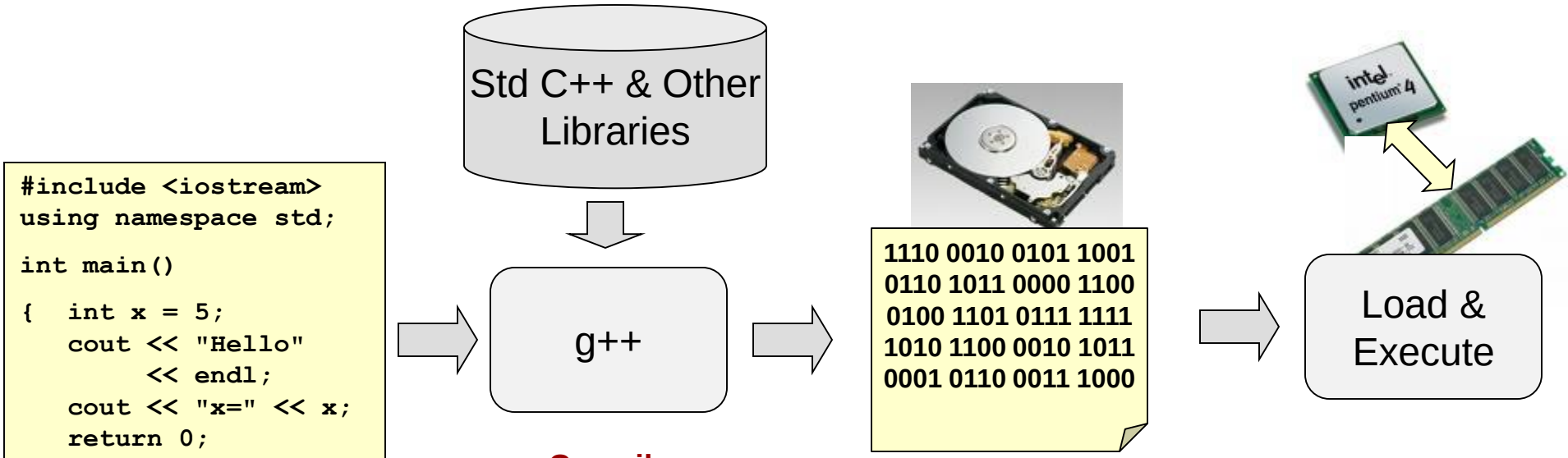
```
$ gedit test.cpp &
$ g++ -g -Wall -o test test.cpp
or
$ make test
```

2 **Compile & fix compiler errors**

```
$ gedit test.cpp &
$ g++ -g -Wall -o test test.cpp
$ ./test
```

3 **Load & run the executable program**

Software Process



C++ file(s) (test.cpp)

-g = Enable Debugging
-Wall =Show all warnings
-o test = Specify Output executable name

Executable Binary Image (test)

```
$ gedit test.cpp &
```

1 Edit & write code

```
$ gedit test.cpp &
$ g++ -g -Wall -o test test.cpp
or
$ make test
```

2 Compile & errors

Fix compile-time errors w/ a debugger

```
$ gedit test.cpp &
$ g++ -g -Wall -o test test.cpp
$ ./test
```

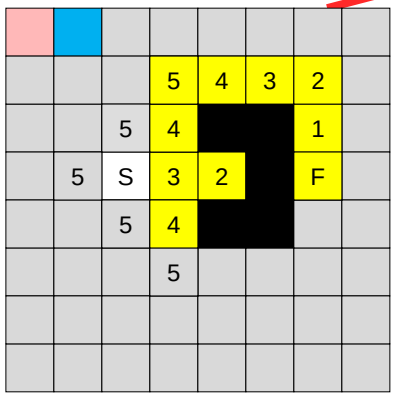
3 Load & run th executable program

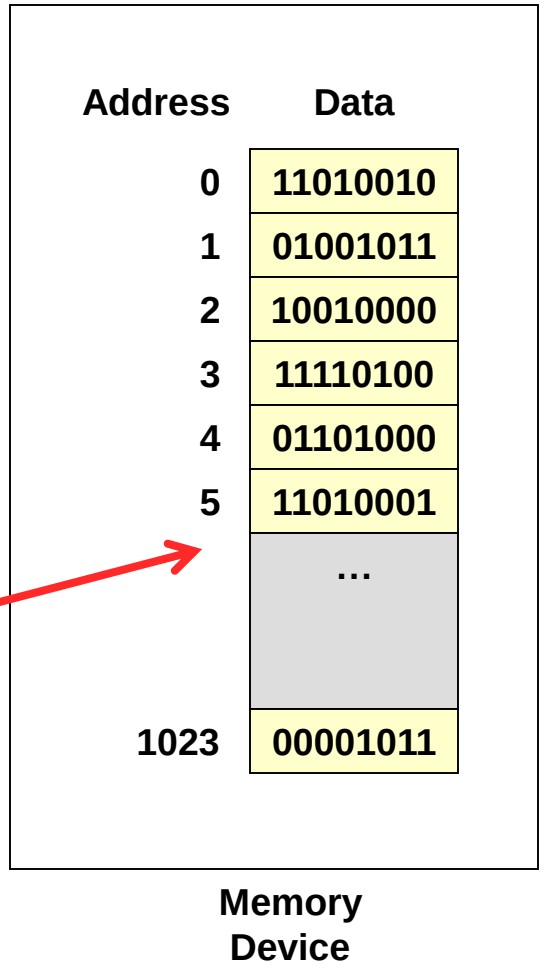
Fix run-time errors w/ a debugger

DATA REPRESENTATION

Memory

- Recall all information in a computer is stored in memory
- Memory consists of cells that each store a group of bits (usually, 1 byte = 8 bits)
- Unique address assigned to each cell
  - Used to reference the value in that location
- We first need to understand the various ways our program can represent data and allocate memory





Starting With Numbers

- A single **bit** can only represent 1 and 0
- To represent more than just 2 values we need to use combinations/sequences of many bits
  - A **byte** is defined as a group 8-bits
  - A **word** varies in size but is usually 32-bits
- So how do we interpret those sequences of bits?
  - Let's learn about number systems

1

A bit

01000001

A byte

0101110 11010001 10110101 01110111

A word

Binary Number System

- Humans use the decimal number system
  - Based on number 10
  - 10 digits: [0-9]
- Because computer hardware uses digital signals with 2 values, computers use the binary number system
  - Based on number 2
  - 2 binary digits (a.k.a bits): [0,1]

Number System Theory

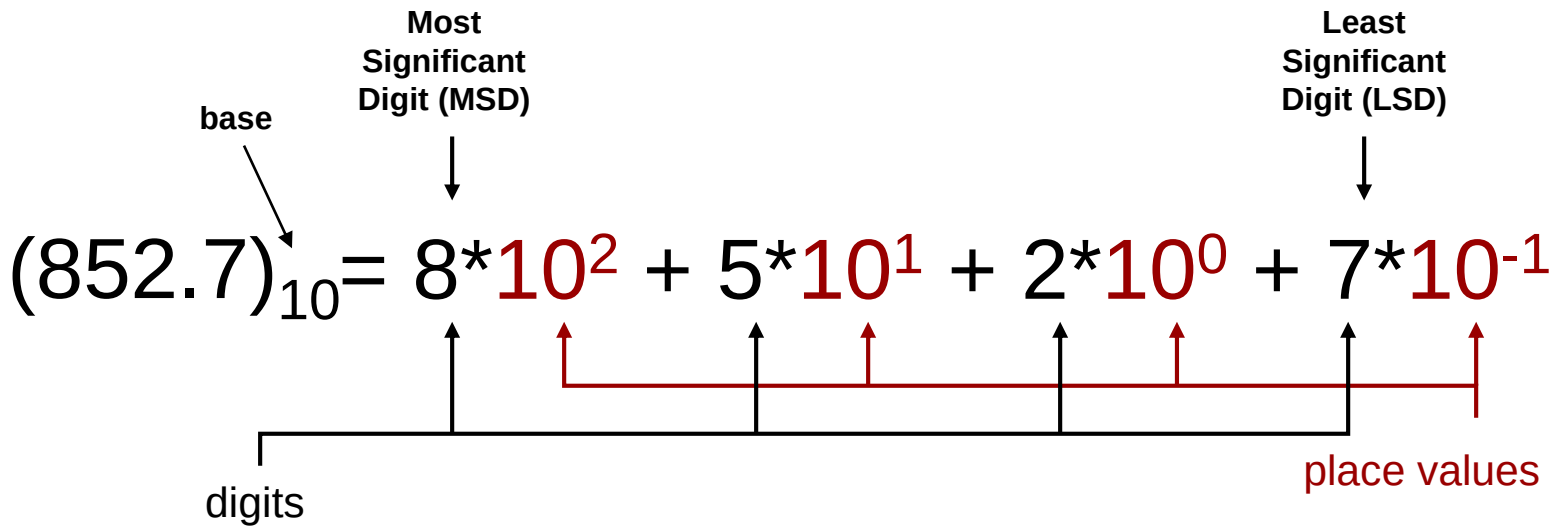
- Let's understand how number systems work by examining decimal and then moving to binary
- The written digits have implied place values
- Place values are powers of the base (decimal = 10)
- Place value of digit to left of decimal point is 10^0 and ascend from there, negative powers of 10 to the right of the decimal point
- The value of the number is the sum of each digit times its implied place value

base

$$(852.7)_{10} =$$

Number System Theory

- The written digits have implied place values
- Place values are powers of the base (decimal = 10)
- Place value of digit to left of decimal point is 10^0 and ascend from there, negative powers of 10 to the right of the decimal point
- The value of the number is the sum of each digit times its implied place value



Binary Number System

- Place values are powers of 2
- The value of the number is the sum of each bit times its implied place value (power of 2)

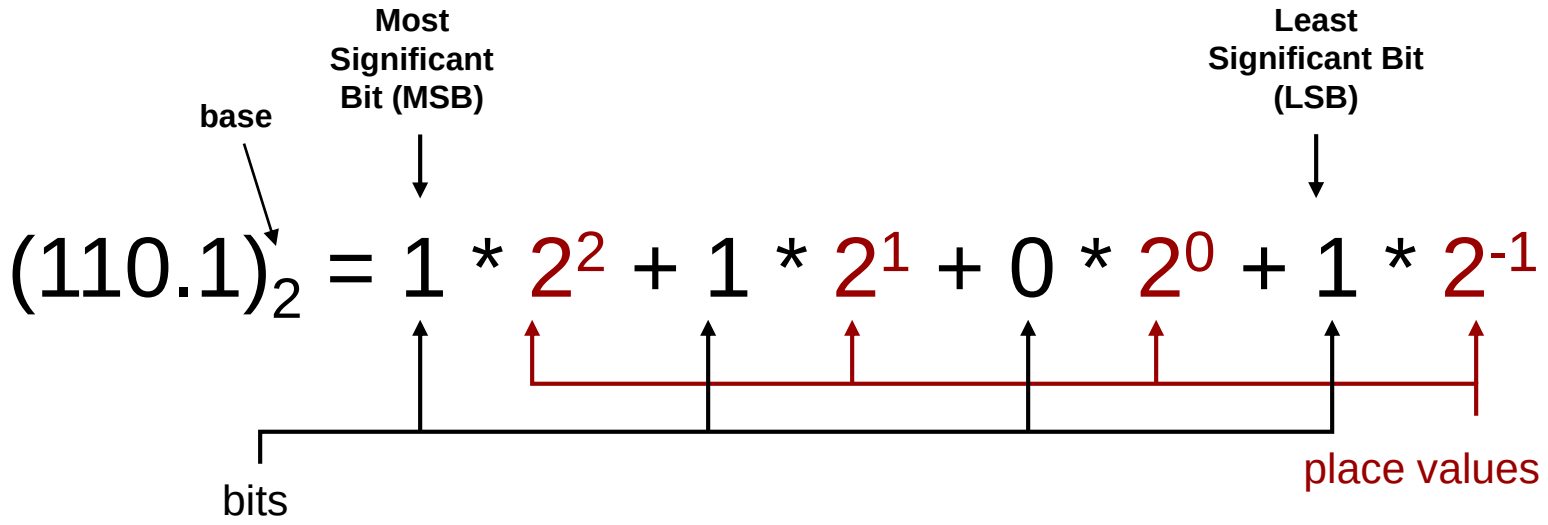
base

$(110.1)_2$ =

$(11010)_2$ =

Binary Number System

- Place values are powers of 2
- The value of the number is the sum of each bit times its implied place value (power of 2)



$(110.1)_2 = 1*4 + 1*2 + 1*.5 = 4+2+.5 = 6.5_{10}$

$(11010)_2 = 1*2^4 + 1*2^3 + 1*2^1 = 26_{10}$

Unique Combinations

- Given *n* digits of base *r*, how many unique numbers can be formed?

2-digit, decimal numbers ____ ____ ___ combinations: ___ - ___
0-9 0-9

3-digit, decimal numbers ____ ____ ____ ____ combinations: ____ - ____

4-bit, binary numbers ____ ____ ____ ____ ___ combinations: _____ - _____
0-1 0-1 0-1 0-1

6-bit, binary numbers ___ ___ ____ ____ ____ ____ ___ combinations: _____ - _____

Main Point: Given n digits of base r, ___ unique numbers can be made with the range [_____]

Sign

- Is there any limitation if we only use the powers of some base as our weights?
  - Can't make negative numbers
- What if we change things
  - How do humans represent negative numbers?
  - Can we do something similar?

	512	256	128	64	32	16	8	4	2	1

1024	512	256	128	64	32	16	8	4	2	1

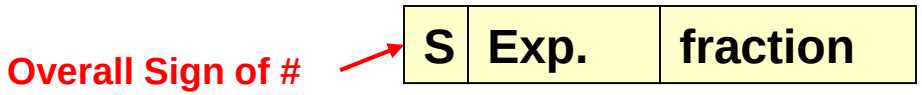
C Integer Data Types

- In C/C++ constants & variables can be of different types and sizes
  - A Type indicates how to interpret the bits and how much memory to allocate
  - Integer Types (signed by default… unsigned with optional leading keyword)

C Type	Bytes	Bits	Signed Range	Unsigned Range
[unsigned] char	1	8	-128 to +127	0 to 255
[unsigned] short	2	16	-32768 to +32767	0 to 65535
[unsigned] long **[unsigned] int**	4	32	-2 billion to +2 billion	0 to 4 billion
[unsigned] long long	8	64	$-8*10^{18}$ to $+8*10^{18}$	0 to $16*10^{18}$

What About Rational/Real #'s

- Previous binary system assumed binary point was fixed at the far right of the number
  - 10010. *(implied binary point)*
- Consider scientific notation:
  - Avogadro's Number: +6.0247 * 10^{23}
  - Planck's Constant: +6.6254 * 10^{-27}
- Can one representation scheme represent such a wide range?
  - Yes! Floating Point
  - Represents the sign, significant digits (fraction), exponent as separate bit fields
- Decimal: ±D.DDD * 10 $^{\pm exp}$
- Binary: ±b.bbbb * $2^{\pm exp}$



C Floating Point Types

- `float` and `double` types:

  Allow decimal representation (e.g. 6.125) as well as very large integers (+6.023E23)

C Type	Bytes	Bits	Range
float	4	32	±7 significant digits * $10^{+/-38}$
double	8	64	±16 significant digits * $10^{+/-308}$

Text

- Text characters are usually represented with some kind of binary code (mapping of character to a binary number such as 'a' = 01100001 bin = 97 dec)
- ASCII = Traditionally an 8-bit code
  - How many combinations (i.e. characters)?
  - English only
- UNICODE = 16-bit code
  - How many combinations?
  - Most languages w/ an alphabet
- In C/C++ a single printing/text character must appear between single-quotes (')
  - Example: 'a', '!', 'Z'

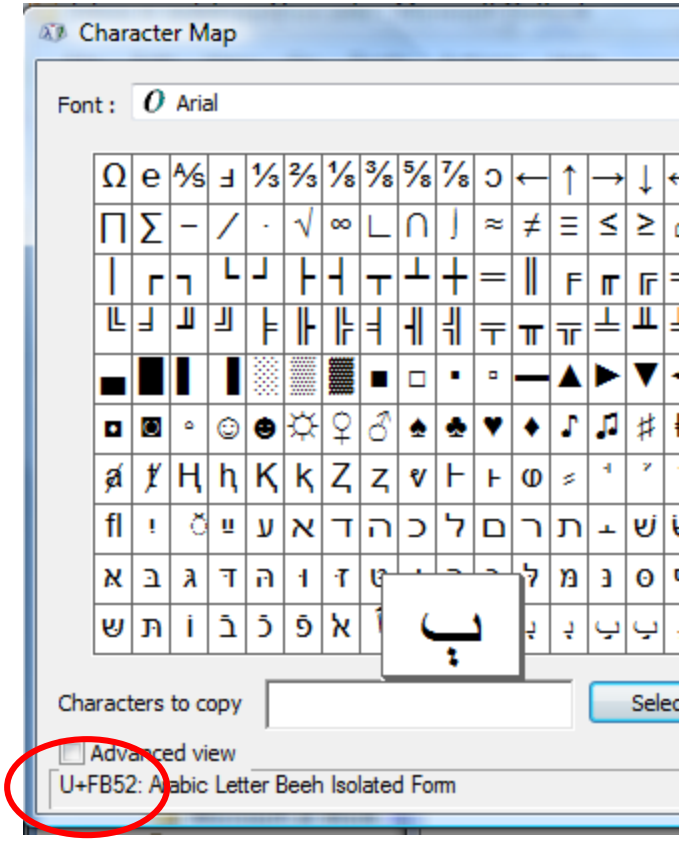
ASCII printable characters

32	space	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	|
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	_		

http://www.theasciicode.com.ar/

UniCode

- ASCII can represent only the English alphabet, decimal digits, and punctuation
  - 7-bit code => 2^7 = 128 characters
  - It would be nice to have one code that represented more alphabets/characters for common languages used around the world
- Unicode
  - 16-bit Code => 65,536 characters
  - Represents many languages alphabets and characters
  - Used by Java as standard character code
  - Won't be used in our course



Unicode hex value
(i.e. FB52 => 1111101101010010)

Interpreting Binary Strings

- Given a string of 1's and 0's, you need to know the *representation system* being used, before you can understand the value of those 1's and 0's.
- Information (value) = Bits + Context (System)

01000001 = ?

Unsigned Binary system

65_{10}

BCD System

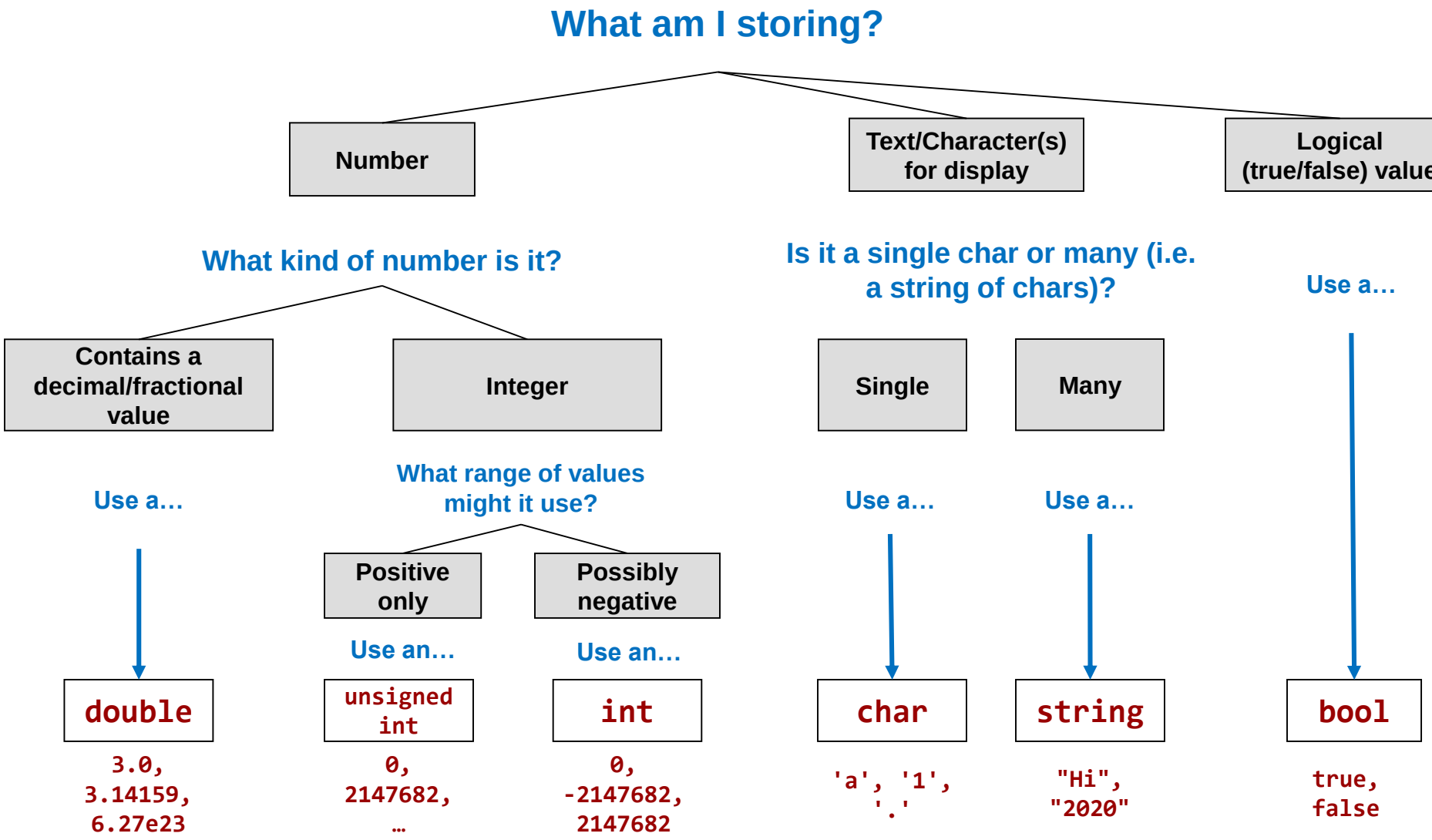
41_{BCD}

ASCII system

'A'$_{ASCII}$

C CONSTANTS & DATA TYPES

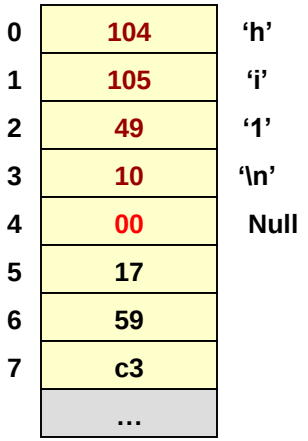
What's Your Type



Constants

- Integer: `496, 10005, -234`
- Double: `12.0, -16., 0.23, -2.5E-1, 4e-2`
- Float: `12.0F` // F = float vs. double
- Characters appear in single quotes
  - `'a', '5', 'B', '!', '\n', '\t', '\\', '\''`
  - Non-printing special characters use "escape" sequence (i.e. preceded by a \ )
  - `'\n' = newline/enter, '\t' = tab`
- C-Strings
  - Multiple characters between double quotes

    `"hi1\n", "12345\n", "b", "\tAns. is %d"`
  - Ends with a `'\0'`=NULL character added as the last byte/character
- Boolean (C++ only): `true, false`
  - Physical representation: 0 = false, (!= 0) = true



String Example

(Memory Layout)

You're Just My Type

- Indicate which constants are matched with the correct type.

Constant	Type	Right / Wrong
4.0	int	
5	int	
'a'	string	
"abc"	string	
5.	double	
5	char	
"5.0"	double	
'5'	int	

You're Just My Type

- Indicate which constants are matched with the correct type.

Constant	Type	Right / Wrong
4.0	int	double (.0)
5	int	int
'a'	string	char
"abc"	string	string (char * or char [])
5.	double	float/double (. = non-integer)
5	char	Int…but if you store 5 in a char variable it'd be okay
"5.0"	double	string (char * or char [])
'5'	int	char

EXPRESSIONS & VARIABLES

Arithmetic Operators

- Addition, subtraction, multiplication work as expected for both integer and floating point types
- Division works 'differently' for integer vs. doubles/floats
- Modulus is only defined for integers

Operator	Name	Example
+	Addition	2 + 5
-	Subtraction	41 - 32
*	Multiplication	4.23 * 3.1e-2
/	Division (Integer vs. Double division)	10 / 3 (=3) 10.0 / 3 (=3.3333)
%	Modulus (remainder) [for integers only]	17 % 5 (result will be 2)

Precedence

- Order of operations/ evaluation of an expression
- Top Priority = highest (done first)
- Notice operations with the same level or precedence usually are evaluated left to right (explained at bottom)
- Evaluate:
  - `2*-4-3+5/2;`
- Tips:
  - Use parenthesis to add clarity
  - Add a space between literals `(2 * -4) - 3 + (5 / 2)`

Operators (grouped by precedence)

struct member operator	*name*`.`*member*
struct member through pointer	*pointer*`->`*member*
increment, decrement	`++, --`
plus, minus, logical not, bitwise not	`+, -, !, ~`
indirection via pointer, address of object	`*`*pointer*`, &`*name*
cast expression to type	`(`*type*`)` *expr*
size of an object	`sizeof`
multiply, divide, modulus (remainder)	`*, /, %`
add, subtract	`+, -`
left, right shift [bit ops]	`<<, >>`
relational comparisons	`>, >=, <, <=`
equality comparisons	`==, !=`
and [bit op]	`&`
exclusive or [bit op]	`^`
or (inclusive) [bit op]	`|`
logical and	`&&`
logical or	`||`
conditional expression	$expr_1$ `?` $expr_2$ `:` $expr_3$
assignment operators	`+=, -=, *=, ...`
expression evaluation separator	`,`

Unary operators, conditional expression and assignment operators group right to left; all others group left to right.

Exercise Review

- Evaluate the following:
  - `25 / 3`
  - `20 - 12 / 4 * 2`
  - `33 % 7`
  - `3 - 5 % 7`
  - `18.0 / 4`
  - `28 - 5 / 2.0`
  - `17 + 5 % 2 - 3`

Exercises from: D.S. Malik, C++ Programming, 5th Ed., Ch. 2, Q6.

C/C++ Variables

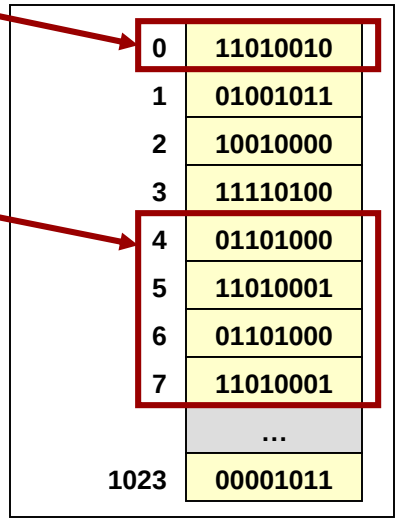
- A computer program needs to operate on and store *data* values (which are usually inputted from the user)
- Variables are just memory locations that are reserved to store a piece of data of specific size and type
- Programmer indicates what variables they want when they write their code
  - Difference: C requires declaring all variables at the beginning of a function before any operations. C++ relaxes this requirement.
- The computer will allocate memory for those variables when the program reaches the declaration

```
#include <iostream>
using namespace std;
int main(int argc, char *argv[])
{
   char c;
   int feet = 50;
   ...
   int inches = 12 * feet;
}
```

Variables must be declared before being used.

char c;
A single-byte variable

int x = 1564983;
A four-byte variable



Variables are actually allocated in RAM when the program is run

C/C++ Variables

- Variables have a:
  - type [`int`, `char`, `unsigned int`, `float`, `double`, etc.]
  - name/identifier that the programmer will use to reference the value in that memory location [e.g. `x`, `myVariable`, `num_dozens`, etc.]
    - Identifiers must start with [A-Z, a-z, or an underscore '_'] and can then contain any alphanumeric character [0-9A-Za-z] (but no punctuation other than underscores)
    - Use descriptive names(e.g. numStudents, doneFlag)
    - Avoid cryptic names (myvar1, a_thing)
  - location [the address in memory where it is allocated]
  - Value
- Reminder: You must declare a variable before using it

What's in a name?

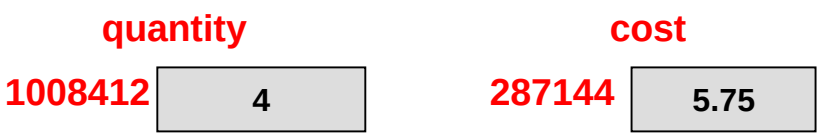
To give descriptive names we often need to use more than 1 word/term. But we can't use spaces in our identifier names. Thus, most programmers use either camel-case or snake-case to write compound names

Camel case: Capitalize the first letter of each word (with the possible exception of the first word)
myVariable, isHighEnough

Snake case: Separate each word with an underscore '_'
my_variable, is_high_enough

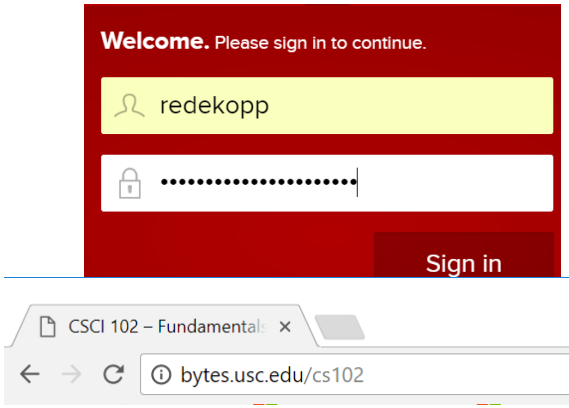
Code

```
int quantity = 4;
double cost = 5.75;
cout << quantity*cost << endl;
```



When To Introduce a Variable

- When a value will be supplied and/or change at run-time (as the program executes)



- When a value is computed/updated at one time and used (many times) later

	A	B
1		
2		80
3		74
4		91
5		83
6		89
7		78
8	SUM	

- To make the code more readable by another human

```
double area = (56+34) * (81*6.25);
// readability of above vs. below
double height = 56 + 34;
double width =  81 * 6.25;
double area = height * width;
```

Assignment operator '='

- Syntax:

```
variable = expression;
```

(LHS) ⟵ (RHS)

  - LHS = Left Hand-Side, RHS = Right Hand Side
- Should be read: Place the value of *expression* into memory location of *variable*
  - `z = x + y - (2*z);`
  - Evaluate RHS first, then place the result into the variable on the LHS
  - If variable is on both sides, we use the old/current value of the variable on the RHS
- **Note**: Without assignment values are computed and then forgotten
  - `x + 5;` // will take x's value add 5 but NOT update x (just throws the result away)
  - `x = x + 5;` // will actually updated x (i.e. requires an assignment)
- Shorthand assignment operators for updating a variable based on its current value: `+=, -=, *=, /=, &=, …`
  - `x += 5;` (x = x+5)
  - `y *= x;` (y = y*x)

Evaluate 5 + 3/2

- The answer is 6.5 ??

Casting

- To achieve the correct answer for 5 + 3 / 2
- Could make everything a double
  - Write 5.0 + 3.0 / 2.0 [explicitly use doubles]
- Could use **implicit** casting (mixed expression)
  - Could just write 5 + 3.0 / 2
    - If operator is applied to mixed type inputs, less expressive type is automatically promoted to more expressive (int is promoted to double)
- Could use C or C++ syntax for **explicit** casting
  - `5 + (double) 3 / (double) 2` (C-Style cast)
  - `5 + static_cast<double>(3) / static_cast<double>(2)` (C++-Style)
  - `5 + static_cast<double>(3) / 2` (cast one & rely on implicit cast of the other)
  - This looks like a lot of typing compared to just writing 5 + 3.0 / 2…but what if instead of constants we have variables
  - `int x=5, y=3, z=2;  x + y/z;`
  - `x + static_cast<double>(y) / z`

Understanding ASCII and chars

- Characters can still be treated as numbers

```
char c = 'a';       // same as  char c = 97;
char d = 'a' + 1;   // c now contains 'b' = 98;
cout << d << endl; // I will see 'b' on the screen

char c = '1';       // c contains decimal 49, not 1
                    // i.e. '1' not equal to 1

c >= 'a' && c <= 'z'; // && means AND
                      // here we are checking if c
                      // contains a lower case letter
```

char c

97

ASCII printable characters

32	space	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	|
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	_		

I/O Streams

- I/O is placed in temporary buffers/streams by the OS/C++ libraries
- cin goes and gets data from the input stream (skipping over preceding whitespace then stopping at following whitespace)
- cout puts data into the output stream for display by the OS (a flush forces the OS to display the contents immediately)

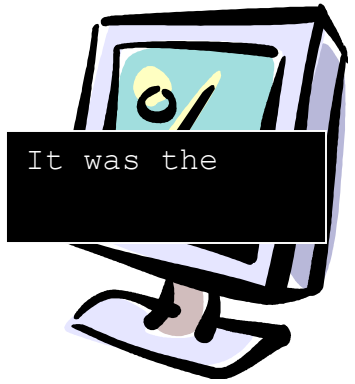
7	5		y	...

input stream:

```
#include<iostream>
using namespace std;
int main()
{
  cout << "It was the" << endl;
  cout << "best of times.";
}
```

output stream:

I	t		w	a	s		t	h	e	\n	b




```
#include<iostream>
using namespace std;
int main()
{
  int x;
  cin >> x;
  return 0;
}
```

input stream:

	y	...

output stream:

4

C++ Output

- Include `<iostream>` (not iostream.h)
- Add `using namespace std;` at top of file
- `cout` (character output) object used to print to the monitor
  - Use the << operator to separate any number of variables or constants you want printed
  - Compiler uses the implied type of the variable to determine how to print it out
  - `endl` constant can be used for the newline character (`'\n'`) though you can still use `'\n'` as well.
    - `endl` also 'flushes' the buffer/stream (forces the OS to show the text on the screen) which can be important it many contexts.

```
#include<iostream>
using namespace std;

int main(int argc, char *argv[])
{
  int x = 5;
  char c = 'Y';
  double y = 4.5;

  cout << "Hello world" << endl;
  cout << "x = " << x << " c = ";
  cout << c << "\ny is " << y << endl;
  return 0;
}
```

Output from program:

```
Hello world
x = 5 c = Y
y is 4.5
```

C++ Input

- 'cin' (character input) object used to accept input from the user and write the value into a variable
  - Use the '>>' operator to separate any number of variables or constants you want to read in
  - Every '>>' means will skip over any leading whitespace looking for text it can convert to the variable form, then stop at the trailing whitespace

```
#include <iostream>
#include <string>
using namespace std;
int main(int argc, char *argv[])
{
  int x;
  char c;
  string mystr;
  double y;

  cout << "Enter an integer, character,
string, and double separated by
spaces:" << endl;

  cin >> x >> c >> mystr >> y;

  cout << "x = " << x << " c = ";
  cout << c << "mystr is " << mystr;
  cout << "y is " << y << endl;
  return 0;
}
```

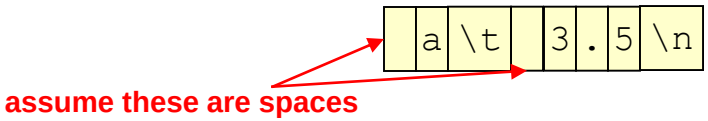
Output from program:

```
Enter an integer, character, string, and double separated by spaces:
5 Y hi 4.5
x = 5 c = Y mystr is hi y is 4.5
```

cin

myc = `0` **y =** `0.0`

- If the user types in



- After the first '>>'

myc = `'a'` **y =** `0.0`

`\t` `3` `.` `5` `\n`

- After the second '>>'

myc = `'a'` **y =** `3.5`

`\n`

```
#include<iostream>
using namespace std;

int main()
{
  char myc = 0;
  double y = 0.0;

  cin >> myc >> y;
  // use the variables somehow...
  return 0;
}
```

Cin…
skips leading whitespace;
stops at trailing whitespace.

In-Class Exercises

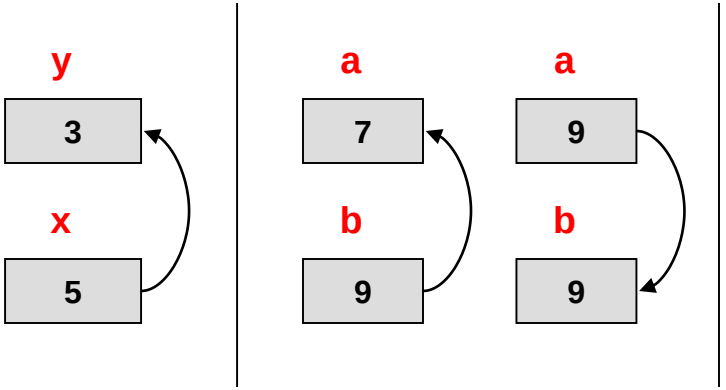
- maxplus
- char_arith

LECTURE 2 / LECTURE 3 END POINT

More Assignments

- Assigning a variable makes a copy
- Challenge: Make a copy

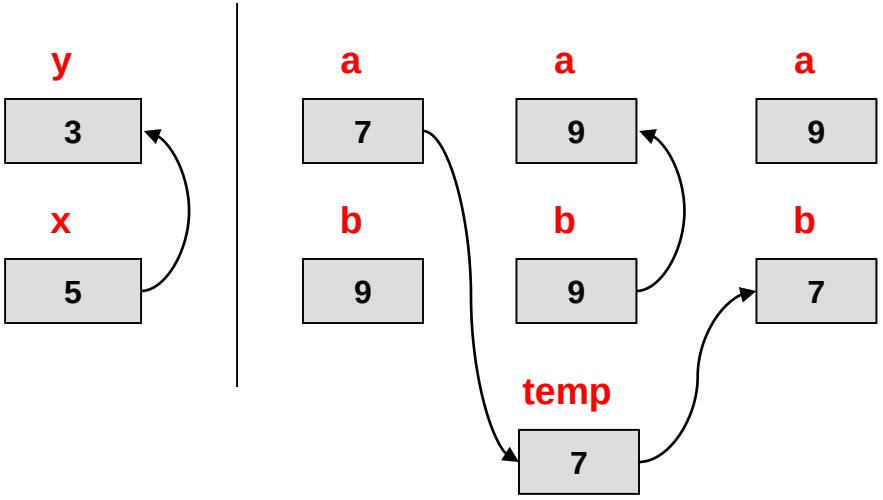
```
int main()
{
  int x = 5, y = 3;
  x = y;    // copy y into x

  // now consider swapping
  // the value of 2 variables
  int a = 7, b = 9;
  a = b;
  b = a;


  return 0;
}
```



More Assignments

- Assigning a variable makes a copy
- Challenge: Make a copy
  - Easiest method: Use a 3rd temporary variable to save one value and then replace that variable

```
int main()
{
  int x = 5, y = 3;
  x = y;    // copy y into x

  // let's try again
  int a = 7, b = 9, temp;
  temp = a;
  a = b;
  b = temp;


  return 0;
}
```



Function call statements

- C++ predefines a variety of functions for you. Here are a few of them:
  - sqrt(x): returns the square root of x (in <cmath>)
  - pow(x, y): returns xy, or x to the power y (in <cmath>)
  - sin(x): returns the sine of x if x is in radians (in <cmath>)
  - abs(x): returns the absolute value of x (in <cstdlib>)
  - max(x, y): returns the maximum of x and y (in <algorithm>)
  - min(x, y): returns the maximum of x and y (in <algorithm>)
- You call these by writing them similarly to how you would use a function in mathematics:

```
#include <iostream>
#include <cmath>
#include <algorithm>
using namespace std;

int main(int argc, char *argv[])
{
  // can call functions
  //  in an assignment
  double res = cos(0);

  // can call functions in an
  //  expression
  res =  sqrt(2) + 2.3 << endl;

  // can call them as part of a
  //  output statement
  cout << max(34, 56) << endl;

  return 0;
}
```

Statements

- End with a semicolon ';'
- Assignment (use initial conditions of **int x=3; int y;**)
  - x = x * 5 / 9; // compute the expression & place result in x
    // x = (3*5)/9 = 15/9 = 1
- Function Call
  - sin(3.14); // Beware of just calling a function w/o assignment
  - x = cos(0.0);
- Mixture of assignments, expressions and/or function calls
  - x = x * y – 5 + max(5,9);
- Return statement (immediate ends a function)
  - return x+y;

In-Class Exercise

- 4swap
- funccall
- hello

In-Class Exercises

- http://bits.usc.edu/cs103/in-class-exercises
  - Hello
  - Tacos
  - Quadratic
  - Math

A Few Odds and Ends

- Comments
  - Anywhere in the code
  - C-Style => "/*" and "*/"
  - C++ Style => "//"
- Variable
  - When declared they will have "garbage" (random or unknown) values unless you initialize them
  - Each variable must be initialized separately
- Scope
  - Global variables are visible to ***all*** the code/functions in the program and are declared ***outside*** of any function
  - Local variables are declared ***inside*** of a function and are ***only*** visible in that function and ***die*** when the function ends

```
/* Anything between slash-star and
   star-slash is ignored even across
   multiple lines of text or code */
/*----Section 1: Compiler Directives ----*/
#include <iostream>
#include <cmath>
using namespace std;

// Global Variables
int x;  // Anything after "//" is ignored

int add_1 (int input)
{
  return (input + 1);
}

int main(int argc, char *argv[])
{
  // y and z are "local" variables
  int y, z=5; // y is garbage, z is five

  z = add_1(z);
  y = z+1;    // an assignment stmt
  cout << y << endl;
  return 0;
}
```

Pre- and Post-Increment Operators

- ++ and -- operators can be used to "increment-by-1" or "decrement-by-1"
  - If ++ comes before a variable it is call pre-increment; if after, it is called post-increment
  - `x++; // If x was 2 it will be updated to 3 (x = x + 1)`
  - `++x; // Same as above (no difference when not in a larger expression)`
  - `x--; // If x was 2 it will be updated to 1 (x = x – 1)`
  - `--x; // Same as above (no difference when not in a larger expression)`
- Difference between pre- and post- is only evident when used in a larger expression
- Meaning:
  - Pre: Update (inc./dec.) the variable before using it in the expression
  - Post: Use the old value of the variable in the expression then update (inc./dec.) it
- Examples [suppose we start each example with: `int y; int x = 3;`]
  - `y = x++ + 5; // Post-inc.; Use x=3 in expr. then inc. [y=8, x=4]`
  - `y = ++x + 5; // Pre-inc.; Inc. x=4 first, then use in expr. [y=9, x=4]`
  - `y = x-- + 5; // Post-dec.; Use x=3 in expr. then dec. [y=8, x=2]`

Exercise

- Consider the code below
  - `int x=5, y=7, z;`
  - `z = x++ + 3*--y + 2*x;`
- What is the value of x, y, and z after this code executes

Not for lecture presentations

BACKUP

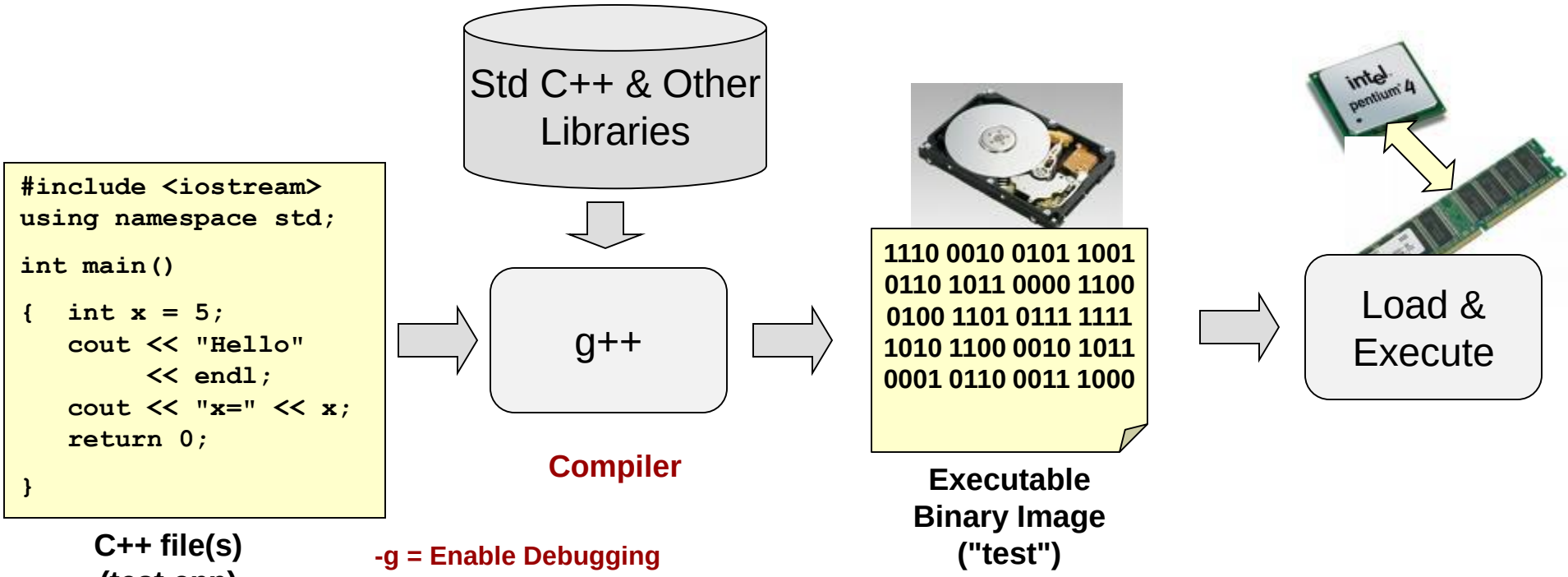
C PROGRAM STRUCTURE AND COMPILATION

C Program Format/Structure

- Comments
  - Anywhere in the code
  - C-Style => "/*" and "*/"
  - C++ Style => "//"
- Compiler Directives
  - #includes tell compiler what other library functions you plan on using
  - 'using namespace std;' -- Just do it for now!
- Global variables (more on this later)
- main() function
  - Starting point of execution for the program
  - Variable declarations often appear at the start of a function
  - All code/statements in C must be inside a function
  - Statements execute one after the next
  - Ends with a 'return' statement
- Other functions

```
/* Anything between slash-star and
   star-slash is ignored even across
   multiple lines of text or code */
/*----Section 1: Compiler Directives ----*/
#include <iostream>
#include <cmath>
using namespace std;

/*------------ Section 2 ----------------*/
/*Global variables & Function Prototypes */

int x;  // Anything after "//" is ignored
void other_unused_function();

/*----Section 3: Function Definitions ---*/
void other_unused_function()
{
  cout << "No one uses me!" << endl;
}

int main(int argc, char *argv[])
{  // anything inside these brackets is
   // part of the main function
  int y;    // a variable declaration stmt
  y = 5+1; // an assignment stmt
  cout << y << endl;
  return 0;
}
```

Software Process



-g = Enable Debugging
-Wall =Show all warnings
-o test = Specify Output executable name

```
$ gedit test.cpp &
```

1 Edit & write code

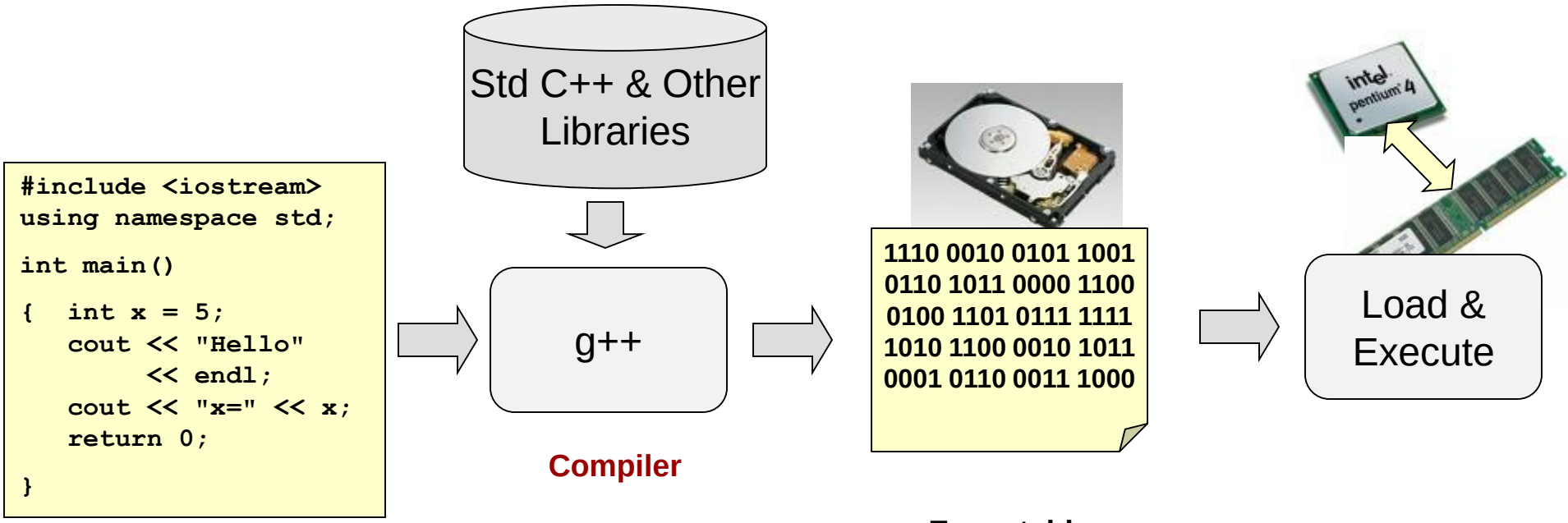
```
$ gedit test.cpp &
$ g++ -g -Wall -o test test.cpp
or
$ make test
```

2 Compile & fix compiler errors

```
$ gedit test.cpp &
$ g++ -g -Wall -o test test.cpp
$ ./test
```

3 Load & run the executable program

Software Process




```
$ gedit test.cpp &
```

1 Edit & write code

```
$ gedit test.cpp &
$ g++ -g -Wall -o test test.cpp
or
$ make test
```

2 Compile & errors

Fix compile-time errors w/ a debugger

```
$ gedit test.cpp &
$ g++ -g -Wall -o test test.cpp
$ ./test
```

3 Load & run th executable program

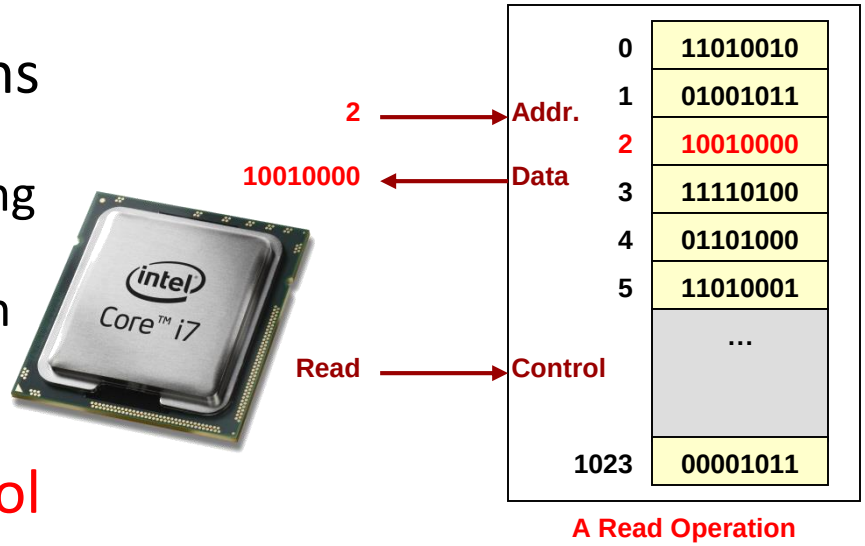
Fix run-time errors w/ a debugger

gdb / ddd / kdbg

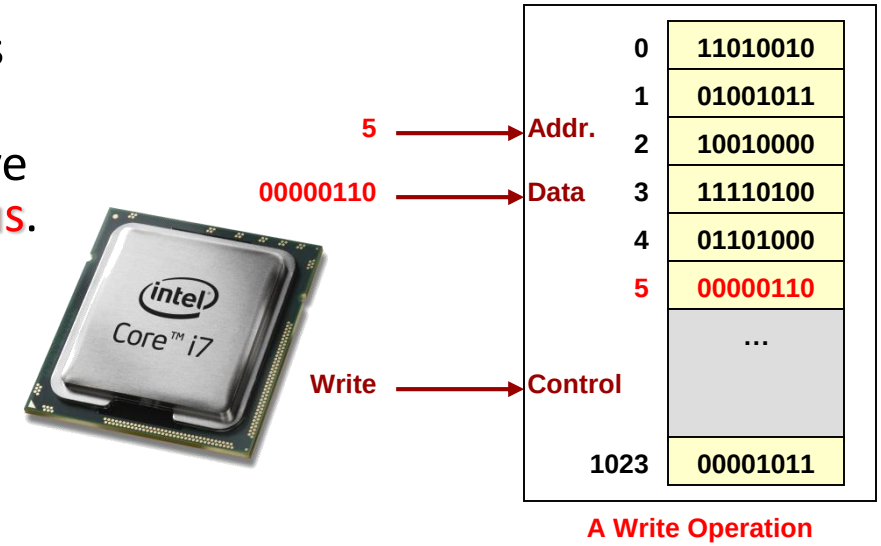
- To debug your program you must have compiled with the '–g' tag in g++ (i.e. g++ –g –Wall –o test test.cpp).
- gdb is the main workhorse of Unix/Linux debuggers (but it is text-based while 'ddd' and 'kdbg' are graphical based debuggers)
  - Run using: $ gdb ./test
- Allows you to…
  - Set breakpoints (a point in the code where your program will be stopped so you can inspect something of interest)
    - 'break 7' will cause the program to halt on line 7
  - Run: Will start the program running until it hits a breakpoint of completes
  - Step: Execute next line of code
  - Next: Like 'Step' but if you are at a function step will go into that function while 'Next' will run the function stopping at the next line of code
  - Print variable values ('print x')

Memory Operations

- Memories perform 2 operations
  - Read: retrieves data value in a particular location (specified using the address)
  - Write: changes data in a location to a new value
- To perform these operations a set of address, data, and control inputs/outputs are used
  - Note: A group of wires/signals is referred to as a 'bus'
  - Thus, we say that memories have an address, data, and control bus.



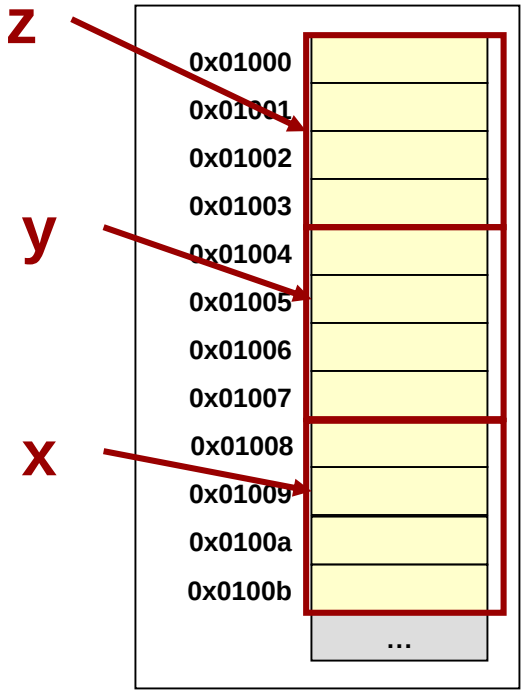
A Read Operation



A Write Operation

Activity 1

- Consider the code below & memory layout
  - `int x=5, y=7, z=1;`
  - `z = x + y - z;`
- Order the memory activities & choose **Read** or **Write**
  1. **R** / **W** value @ addr. 0x01008
  2. Allocate & init. memory for x, y, & z
  3. **Read** value @ addr. 0x01000
  4. **Write** value @ addr. 0x01000
  5. **R** / **W** value @ addr. 0x01004
- Answer: 2, 1(R), 5(R), 3, 4



Memory & corresponding variable allocation