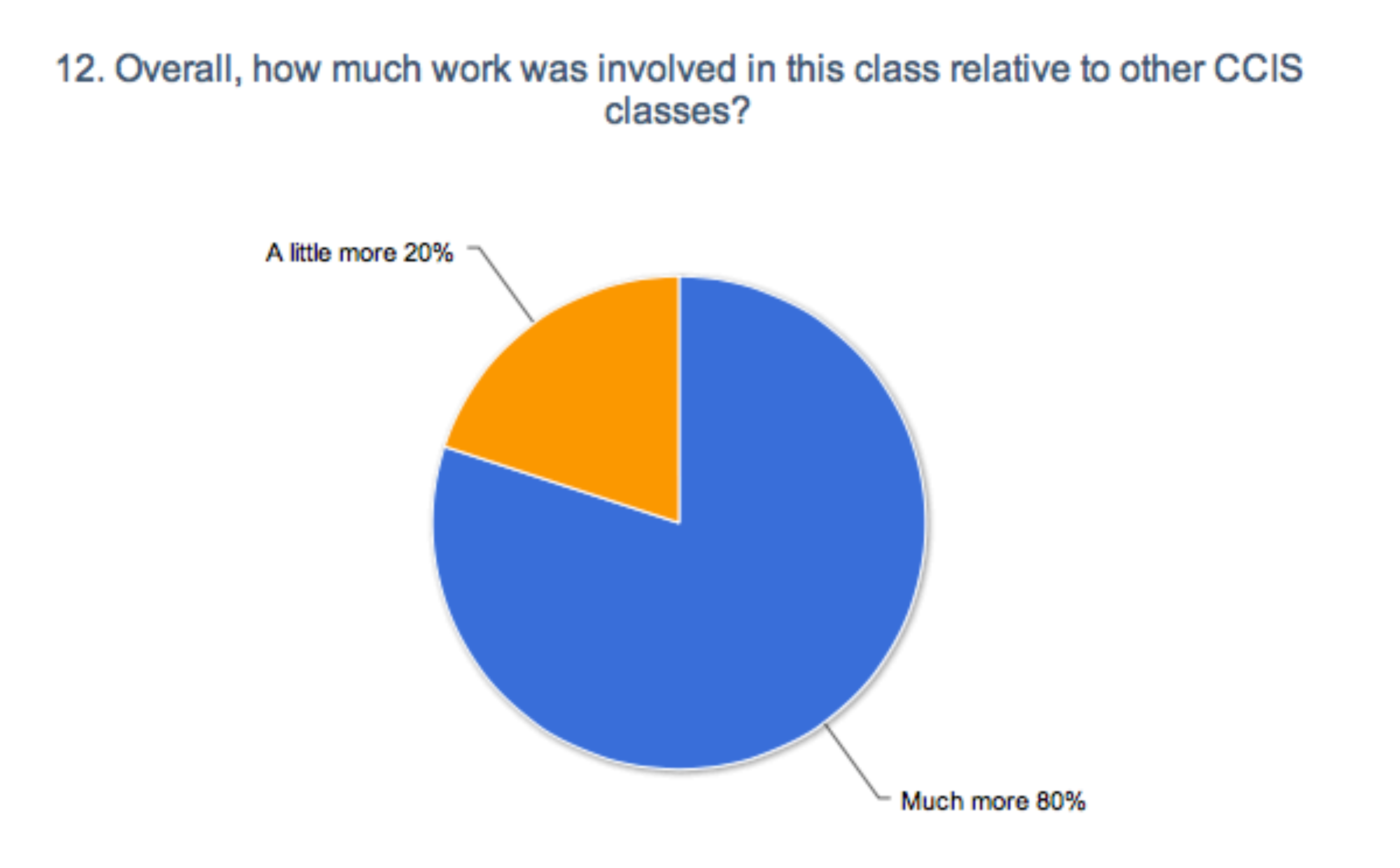
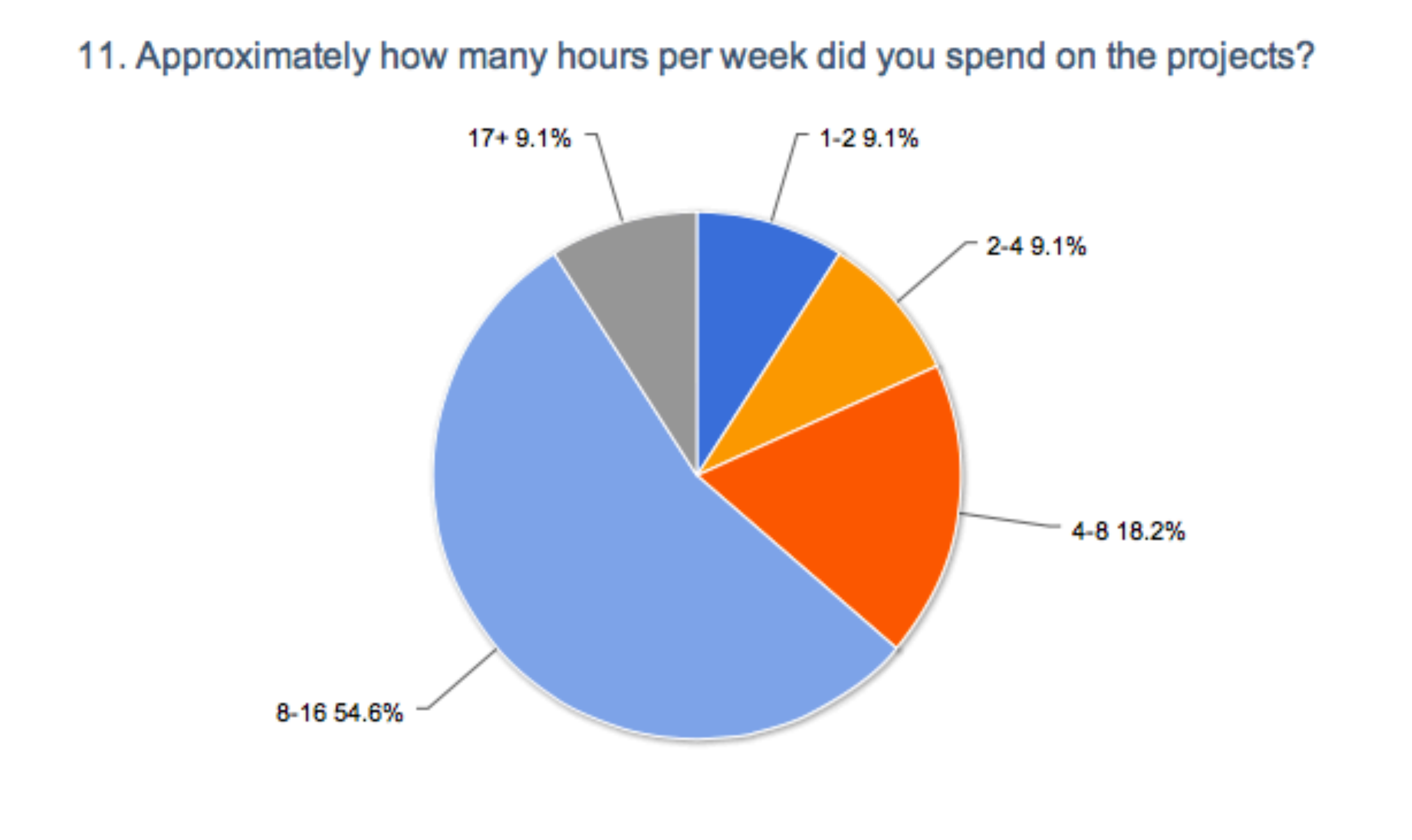
Work relative to other classes



12. Overall, how much work was involved in this class relative to other CCIS classes?

Hours/week on projects



C Bootcamp

Day 1

CS3600, Northeastern University

Alan Mislove

Slides adapted from Anandha Gopalan's CS132 course at Univ. of Pittsburgh

Overview

C: A language written by Brian Kernighan and Dennis Ritchie. This was to be the language that UNIX was written in to become the first "portable" language

C is an typed, imperative, call-by-value language

Programming in C

Four stages

- Editing: Writing the source code by using some IDE or editor
- Preprocessing: Already available routines
- Compiling: Translates source to object code for a specific platform
- Linking: Resolves external references and produces the executable module

For now, we reduce these to two

- Editing (use your favorite editor)
- Compiling (use `make`)

You will also use `make` to test your program

Example C program

Example: Hello World in C

```c
#include <stdio.h>

int main(int argc, char *argv[]) {
  printf("Hello, world!\n");
  return 0;
}
```

Line 1

```
1: #include <stdio.h>
```

As part of compilation, the C compiler runs a program called the C preprocessor. The preprocessor is able to add and remove code from your source file.

In this case, the directive `#include` tells the preprocessor to include code from the file `stdio.h`.

This file contains declarations for functions that the program needs to use. A declaration for the `printf` function is in this file.

Line 3

```
3: int main(int argc, char *argv[]) {
```

This statement declares the `main` function.

A C program can contain many functions but must have one `main`.

A function is a self-contained module of code that can accomplish some task.

The `int` specifies the return type of main. By convention, a status code is returned (`0` represents success, any other value indicates an error).

Line 4

```
4:    printf("Hello, world!\n");
```

`printf` is a function from a standard C library that is used to print strings to the standard output (normally the terminal).

The compiler links code from these standard libraries to the code you have written to produce the final executable.

The `\n` is a special format modifier that tells the `printf` to put a line feed (ASCII character 10) at the end of the line.

If there were another `printf` in this program, its string would print on the next line.

Line 5

```
5:   return 0;
```

Returns `0` from the current function. For the `main` function, this is the return status of the program.

The type of this value must match the function definition

No statements after the `return` will be executed

Data types and operators

Data types

Data type determines

- How it is represented internally by the hardware
- How it may be legally manipulated, (operations allowed on that data type)
- What values it may take on

Constants and variables are classified into 4 basic types

- Character: char (1 byte)
- Integer: int (usually 4 bytes)
- Floating Point: float (usually 4 bytes)
- All other data objects are built up from these fundamental types

char

What is a character

- A member of the character set
- ASCII: American Standard Code for Information Interchange (128 characters)
- Each character is internally represented using a number (e.g: A is 65)

Recognized types, sizes and ranges:

- `char`: 1 byte (0 ≤ x ≤ 255)
- `unsigned char`: 1 byte (0 ≤ x ≤ 255)
- `signed char`: 1 byte (-128 ≤ x ≤ 127)

char (cont.)

ASCII character ‘2’ and the number 2 are not the same (`2 != '2'`)

- `'2'` is the character 2 (50 in ASCII)
- `2` is the number 2
- As a result, `'2' = 50`

Escape Sequence

- Special characters denoted by ‘\’ followed by characters or hexadecimal code

Common sequences

`\n`	new line	`\a`	alert
`\t`	tab	`\\`	backslash
`\r`	carriage return	`\”`	double quote

int, short, long

A whole or integral number, not containing a fractional part

Recognized types, size and ranges:

- `short int`: 2 bytes ($-2^{15} \leq x \leq 2^{15} - 1$)
- `int`: 4 bytes ($-2^{31} \leq x \leq 2^{31} - 1$, signed by default)
- `unisgned int`: 4 bytes ($0 \leq x \leq 232\text{-}1$)
- `long long int`: 8 bytes ($-2^{63} \leq x \leq 2^{63} - 1$, signed by default)

Expressible as octal, decimal or hexadecimal

- Integer starting with `0` will be interpreted in octal (e.g: `010` is 8)
- Integer starting with `0x` will be interpreted in hexadecimal (`0x10` is 16)

Negative numbers typically represented using two's complement

- What does `short a = -7` look like in machine representation?

float, double

A number which may include a fractional part

Representation is an estimate (although at times, it may be exact)

- `2.13` can be represented exactly
- `1.23456789 x 1028` does not fit into a float
  - Hence number is truncated

Recognized types, sizes and ranges

- `float`: 4 bytes (1 sign bit, 8 exponent bits, 24 fraction bits)
  - Range is -1e37 ≤ x ≤ 1e38
- `double`: 8 bytes (1 sign bit, 11 exponent bits, 53 fraction bits)
  - Range is -1e307 ≤ x ≤ 1e308
- `long double`: usually the same as double, sometimes 80 bits

Arithmetic operators

Assignment (=)	`a = b`	Equal to	`a == b`
Addition (+)	`a + b`	Not equal to	`a != b`
Subtraction (-)	`a – b`	Less than	`a < b`
Multiplication (*)	`a * b`	Less than or equal to	`a <= b`
Division (/)	`a / b`	Increment	a++
Modulus (%)	`a % b`	Decrement	a––

Order of precedence (highest to lowest)

- Parenthesis
- Multiplication, division, modulus
- Addition, subtraction
- Comparisons
- Assignment

Bitwise operators

Ints can be viewed as collections of bits; C provides bitwise operations

<table>
<tr><td>Bitwise AND (&)</td><td>a & b</td></tr>
<tr><td>Bitwise OR (|)</td><td>a | b</td></tr>
<tr><td>Bitwise NOT (~)</td><td>~a</td></tr>
<tr><td>Bitwise XOR (^)</td><td>a ^ b</td></tr>
<tr><td>Bitwise left shift (<<)</td><td>a << b</td></tr>
<tr><td>Bitwise right shift (>>)</td><td>a >> b</td></tr>
</table>

Examples:

```
unsigned int a = 9;
unsigned int b = 3;
printf("9 & 3 is: %d\n", (a & b));  // ?
printf("9 << 3 is: %d\n", (a << b));  // ?
```

Constants

Constants

- Constants are values written directly into program statements
  - Not stored in variables
  - Can assign constant values to variables

```
unsigned int x = 122;
```

- Constants never change value
  - Cannot assign value to any type of constant

- Character constants
  - Represented in ‘’ (single quotes)

```
char x = 'c';
```

Numerical constants

Integer constants

- Default data type is `int`
- If value exceeds `int` range, it then becomes `long`
- Can force compiler to store value as long/unsigned using l/u suffixes

```
long c = 0x0304lu;
```

Floating point constants

- Default type of floating point is `double`
- Can force type float during compilation by using decimal (e.g., `1.0`)
- Can also use scientific notation

```
float x = 1.3e-20;
```

Variables

Variables

Variables are placeholders for values

Variable declaration (e.g., `int x = 7;`)

- Associates data type with the variable name
- Allocates memory of proper size and associates variable name with it
- Cannot be re-declared within the scope of prior declaration
- Variable memory space initially contains junk, unless initialized

Variable name

- Set of characters starting with [A-Z] or [a-z] or underscore
- Period (.) is not allowed; no leading numbers; no reserved keywords
- Case sensitive
  - Convention is lowercase used for variables and UPPERCASE for constants

Variable types

Character variables:

Declared using the char keyword

```
char first = 'c';
char second = 100;
```

Integer variables

Declared using the int keyword, with modifiers

```
int a = 8;
short unsigned int c = 2;
```

Floating point variables

Declared using the float or double keyword

```
float g = 3.4;
double e = -2.773;
```

Arrays

A “data structure” storing a collection of identical objects.

- Allocated memory space is contiguous.
- Identified using a single variable
- The size is equal to the number of elements and is a “constant”.

E.g., create an array called `test_array` which has 10 integer elements

```
int test_array[10];
int my_array[2] = { 30, 10 };
```

Array Elements

- Referenced by name ,index (0-indexed; 2nd element is `test_array[1]`)
- Array bounds not checked

```
test_array[123] = 7; // will compile and run, with unpleasant results
```

Commonly used functions

printf

- Contained in the header file `<stdio.h>`

```
int printf (“string of chars and conversion specs”, arg1, arg2, ...);
```

- The string is output to stdout with conversion specs substituted
  - Returns an int, which is the number of bytes written, or EOF on error

```
printf("Hello world\n");

[amislove@joshua]$ ./a.out
Hello World
[amislove@joshua]$
```

```
printf("Hello world");

[amislove@joshua]$ ./a.out
Hello World[amislove@joshua]$
```

printf Conversion specification

Output conversion specification.

`%[-][field_width_min][.][precision][qualifier]conv_character`

Starts with the `%` character and ends with the `conv_character`

- None or more options may appear between `%` and the `conv_character`

`-` left justification in field (default is right)

`field width min` specifies minimum field width, auto expanded

`.` field separator.

`precision` max num characters to right of decimal in float/double

`qualifier` allows us to specify more about the output

- `h` for short, `l` for long, `ll` for long long

printf conversion chars

Determines how printf prints the value

d, i	integers
u	unsigned integer
o	unsigned octal integer
X, x	unsigned hexadecimal integer
c	single character
s	string (more next lecture)
f	floating point
E, e	floating point in scientific notation

```
printf("%d, %x, %c\n", 78, 78, 78);
```

```
[amislove@joshua]$./a.out
78, 0x4e, N
[amislove@joshua]$
```

casting

Ability to cast a datatype to look like another datatype

Operands converted to single type before expression evaluation

- Generally converted to the longest type
  - char is converted to int
  - float is converted to double
  - int is converted to float

Problems with expressions

- Compiler does not know what the user wants to do with that expression
- Default behavior is to force expression to fit the target
- When target is smaller size, then potential loss of accuracy/precision
- When target is larger, then in general, no problem.

Manual casting

Casting by placing desired type in (), preceding the item to be cast

```
y = (int) 3.14 * x;
z = (double) y;
```

Cannot cast everything, only comparable data types

```
y = (char*) 3.14 * x; //does not compile
```

- Cant do this as structure of double and string are different
- Casting is a unary operator with high precedence

Casting does not alter the cast variable or expression

- Only alters the variable/expression value as it is assigned or manipulated

sizeof

Returns number of bytes required to store a specific data type.

Usage: `int x = sizeof(<argument>);`

Works with types, variables, arrays, structures:

Size of an int	`sizeof(int)`
Size of a structure	`sizeof(struct foo)`
Size of an array element:	`sizeof(test_array[0]);`
Size of the entire array:	`sizeof(test_array);`
Size of int variable x:	`sizeof(x);`

Integer division

An integer function that returns an integer

- NOTE: divisor cannot be zero
- Result is always an integer.

```
30 / 2 = 15;
31 / 2 = 15;
```

- Can force float division through casting

```
31 / (float) 2 = 15.5;
```

Modulus operator yields the integer remainder of an integer division

```
30 % 2 = 0;
31 % 2 = 1;
```