Lecture 2:
Variable Types, Input, and Output

Sarah Nadi
nadi@ualberta.ca
Department of Computing Science
University of Alberta

CMPUT 201 - Practical Programming Methodology

[With material/slides from Guohui Lin, Davood Rafei, and Michael Buro. Most examples taken from K.N. King's book]

Agenda

- Macro definitions for constants
- Basic variable types
- printf
- scanf

Readings

- Textbook: Ch 2, Ch 3, Ch 7 (superficially for now)

SSH & Compilation Recap

demo

Computing the Dimensional Weight of a Box

- We want to create a program that outputs the volume and dimensional weight of a box, according to these formulas:

  Volume = length * width * height
  Dimensional weight = volume / 166 (must be rounded up)

- This will be the output of the program:

```
Dimensions: 12 * 10 * 8
Volume (cubic inches): 960
Dimensional weight (pounds): 6
```

Computing the Dimensional Weight of a Box

- We want to create a program that outputs the volume and dimensional weight of a box, according to these formulas:

  Volume = length * width * height
  Dimensional weight = volume / 166 (must be rounded up)

- This will be the output of the program:

```
Dimensions: 12 * 10 * 8
Volume (cubic inches): 960
Dimensional weight (pounds): 6
```

How many variables do we need?

```
/* Computes the dimensional weight of a 12 * 10 * 8 box */

#include <stdio.h>

int main (void){

 int height, length, width, volume, weight;

 height = 8;
 length = 12;
 width = 10;

 volume = height * length * width;
 weight = (volume + 165) / 166;

 // printing?

}
```

Before printing, let's make our program a bit more understandable…

```
/* Computes the dimensional weight of a 12 * 10 * 8 box */

#include <stdio.h>

#define INCHES_PER_POUND 166

int main (void){

  int height, length, width, volume, weight;

  height = 8;
  length = 12;
  width = 10;

  volume = height * length * width;
  weight = (volume + 165) / INCHES_PER_POUND;

  //printing?

}
```

this is called a macro definition. It is a way to define names for constants that will not change throughout the program

The `printf` Function

- Must include `#include <stdio.h>` to use it

- Example:

```
int age = 23;
printf("Your age is %d", age);
```

- Has the signature: `printf(string, expr1, expr2, …);`

- Displays the contents of a **format string**

  - has "placeholders" for values inserted at specific points

  - must indicate one value at a time

  - every "placeholder" needs a **conversion specification that begins with %** and **depends on the type of the variable being printed**

```
/* Computes the dimensional weight of a 12 * 10 * 8 box */

#include <stdio.h>

int main (void){

 int height, length, width, volume, weight;

 height = 8;
 length = 12;
 width = 10;
 volume = height * length * width;
 weight = (volume + 165) / 166;


 printf("Dimensions: %d * %d * %d\n", length, width, height);
 printf("Volume (cubic inches): %d\n", volume);
 printf("Dimensional weight (pounds): %d\n", weight);

}
```

```
/* Computes the dimensional weight of a 12 * 10 * 8 box */

#include <stdio.h>

int main (void){

 int height, length, width, volume, weight;

 height = 8;
 length = 12;
 width = 10;
 volume = height * length * width;
 weight = (volume + 165) / 166;


 printf("Dimensions: %d * %d * %d\n", length, width, height);
 printf("Volume (cubic inches): %d\n", volume);
 printf("Dimensional weight (pounds): %d\n", weight);

}
```

Format string

```
/* Computes the dimensional weight of a 12 * 10 * 8 box */

#include <stdio.h>

int main (void){

 int height, length, width, volume, weight;

 height = 8;
 length = 12;
 width = 10;
 volume = height * length * width;
 weight = (volume + 165) / 166;


 printf("Dimensions: %d * %d * %d\n", length, width, height);
 printf("Volume (cubic inches): %d\n", volume);
 printf("Dimensional weight (pounds): %d\n", weight);

}
```

Format string

conversion specification

```
/* Computes the dimensional weight of a 12 * 10 * 8 box */

#include <stdio.h>

int main (void){

 int height, length, width, volume, weight;

 height = 8;
 length = 12;
 width = 10;
 volume = height * length * width;
 weight = (volume + 165) / 166;


 printf("Dimensions: %d * %d * %d\n", length, width, height);
 printf("Volume (cubic inches): %d\n", volume);
 printf("Dimensional weight (pounds): %d\n", weight);

}
```

Format string

conversion specification

expression

```
/* Computes the dimensional weight of a 12 * 10 * 8 box */

#include <stdio.h>

int main (void){

 int height, length, width, volume, weight;

 height = 8;
 length = 12;
 width = 10;
 volume = height * length * width;
 weight = (volume + 165) / 166;

 printf("Dimensions: %d * %d * %d\n", length, width, height);
 printf("Volume (cubic inches): %d\n", volume);
 printf("Dimensional weight (pounds): %d\n", weight);

}
```

Format string

conversion specification

expression

```
/* Computes the dimensional weight of a 12 * 10 * 8 box */

#include <stdio.h>

int main (void){

 int height, length, width, volume, weight;

 height = 8;
 length = 12;
 width = 10;
 volume = height * length * width;
 weight = (volume + 165) / 166;

 printf("Dimensions: %d * %d * %d\n", length, width, height);
 printf("Volume (cubic inches): %d\n", volume);
 printf("Dimensional weight (pounds): %d\n", weight);

}
```

Format string

conversion specification

expression

```
/* Computes the dimensional weight of a 12 * 10 * 8 box */

#include <stdio.h>

int main (void){

 int height, length, width, volume, weight;

 height = 8;
 length = 12;
 width = 10;
 volume = height * length * width;
 weight = (volume + 165) / 166;

 printf("Dimensions: %d * %d * %d\n", length, width, height);
 printf("Volume (cubic inches): %d\n", volume);
 printf("Dimensional weight (pounds): %d\n", weight);

}
```

Format string

conversion specification

expression

Variable Types

- Integer types (`short int`, `unsigned short int`, `int`, `unsigned int`, `long int`, `unsigned long int`) — whole numbers

- Floating types (`float`, `double`, `long double`) — can have fractional parts

- Character types (`char`)

Printing Different Variable Types

```
int x = 8;
float y = 4.5;
double z = 7.89;
char c = ‘h’;
printf(“%d”, x); // prints 8
printf(“%f”, y); // prints 4.500000
printf(“%f”, z); // prints 7.890000
printf(“%c”, c); // prints h
```

Printing Different Variable Types

```
int x = 8;
float y = 4.5;
double z = 7.89;
char c = ‘h’;
printf(“%d”, x); // prints 8
printf(“%f”, y); // prints 4.500000
printf(“%f”, z); // prints 7.890000
printf(“%c”, c); // prints h
```

Note how the letter that comes after the % sign differs depending on the variable type

`printf` Conversion Specifiers

- Form: *%m.pX* or *%-m.pX*

  - *m* is an optional integer constant that specifies the minimum field width

  - \- is used to specify left justification of the output text

  - *p* is an optional integer constant that indicates precision. If omitted, so is the period

  - *X* is a required letter that depends on the variable type (see next slide)

`printf` Conversion Specifiers

- Choices of *X* in format specifiers (Best way to understand these is to just try them out!):
  - *d:* displays integer in decimal form. *p* here indicates minimum number of digits.
  - *e*: displays floating-point number in exponential form, i.e., scientific notation. *p* indicates no. of digits after decimal pt (default=6). If *p*==0, no decimal pt displayed.
  - *f:* displays a floating-point number in “fixed decimal format”, without an exponent. *p* has same meaning as in *e* above.
  - *g:* displays a floating-point number in either exponential format or fixed decimal format, depending which results in a shorter representation. *p* indicates max. no. of significant digits to be displayed (not digits after decimal point). It doesn’t show trailing 0’s. If value doesn’t have a fraction, it won’t display the decimal point.
  - More will be covered later

demo: tprintf.c

`printf` Conversion Specifiers

- Choices of *X* in format specifiers (Best way to understand these is to just try them out!):
  - *d:* displays integer in decimal form. *p* here indicates minimum number of digits.
  - *e*: displays floating-point number in exponential form, i.e., scientific notation. *p* indicates no. of digits after decimal pt (default=6). If *p*==0, no decimal pt displayed.
  - *f:* displays a floating-point number in “fixed decimal format”, without an exponent. *p* has same meaning as in *e* above.
  - *g:* displays a floating-point number in either exponential format or fixed decimal format, depending which results in a shorter representation. *p* indicates max. no. of significant digits to be displayed (not digits after decimal point). It doesn’t show trailing 0’s. If value doesn’t have a fraction, it won’t display the decimal point.
  - More will be covered later

Quiz using mentimeter

demo: tprintf.c

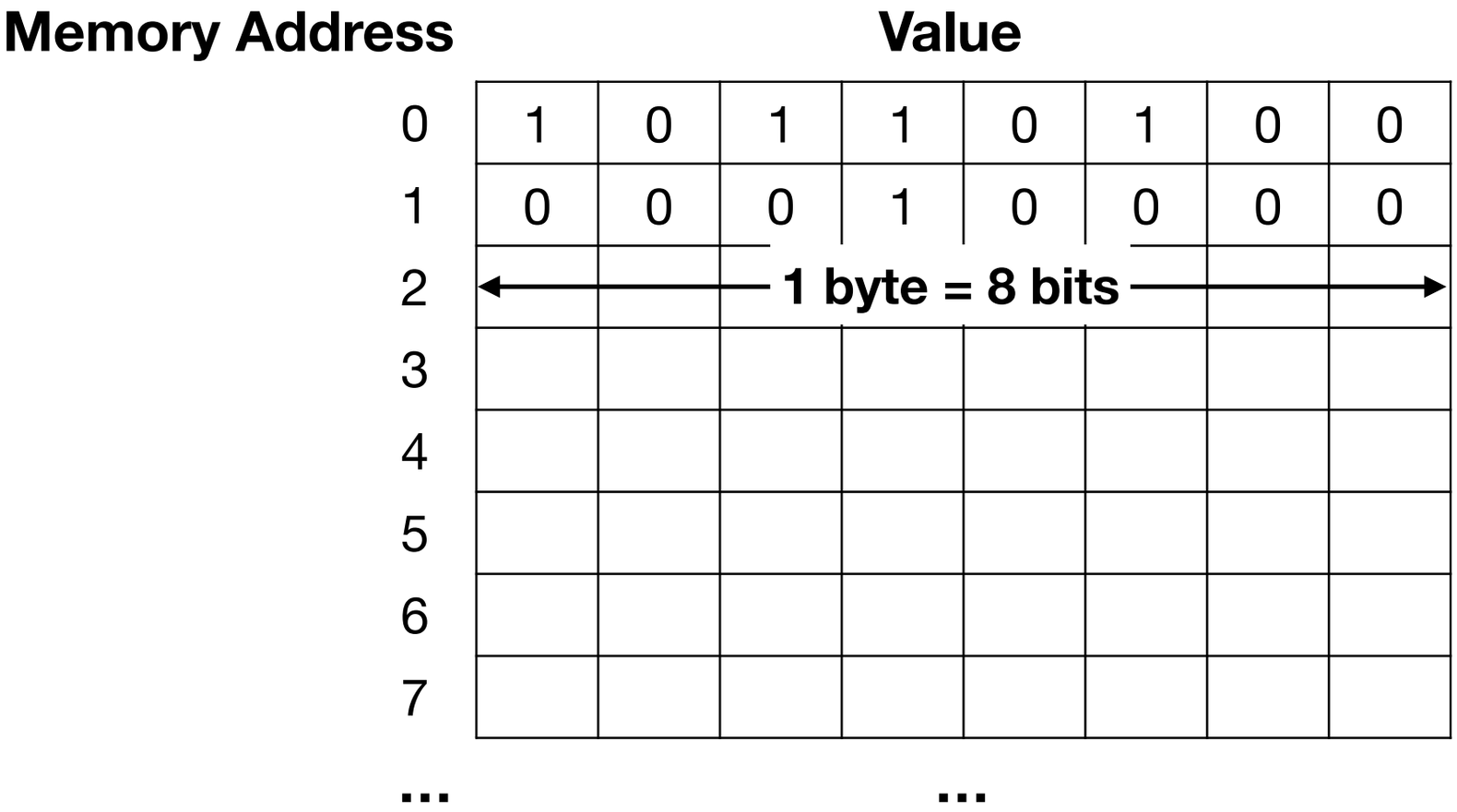
Escape Characters

- Escape sequences start with “\”. Used for:
  - reserved symbols such as “ (use \”)
  - non-printing characters:
    - backspace: \b
    - new line: \n
    - backslash: \\
    - percent: %%
    - horizontal tab: \t
    - alert: \a

demo: special_print.c

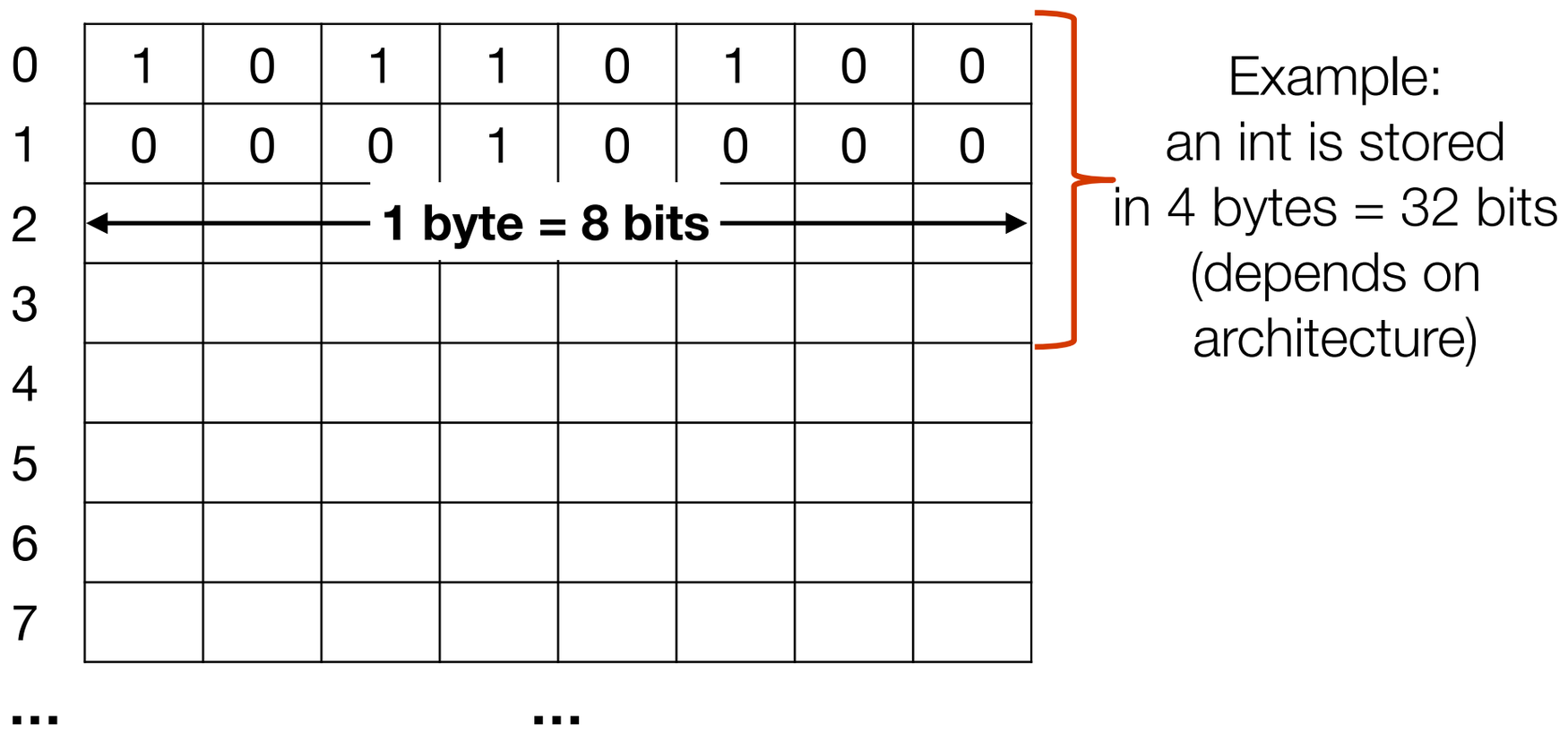
Why are Variable Types Important?

- All variables in your program are stored some place in memory in binary form.



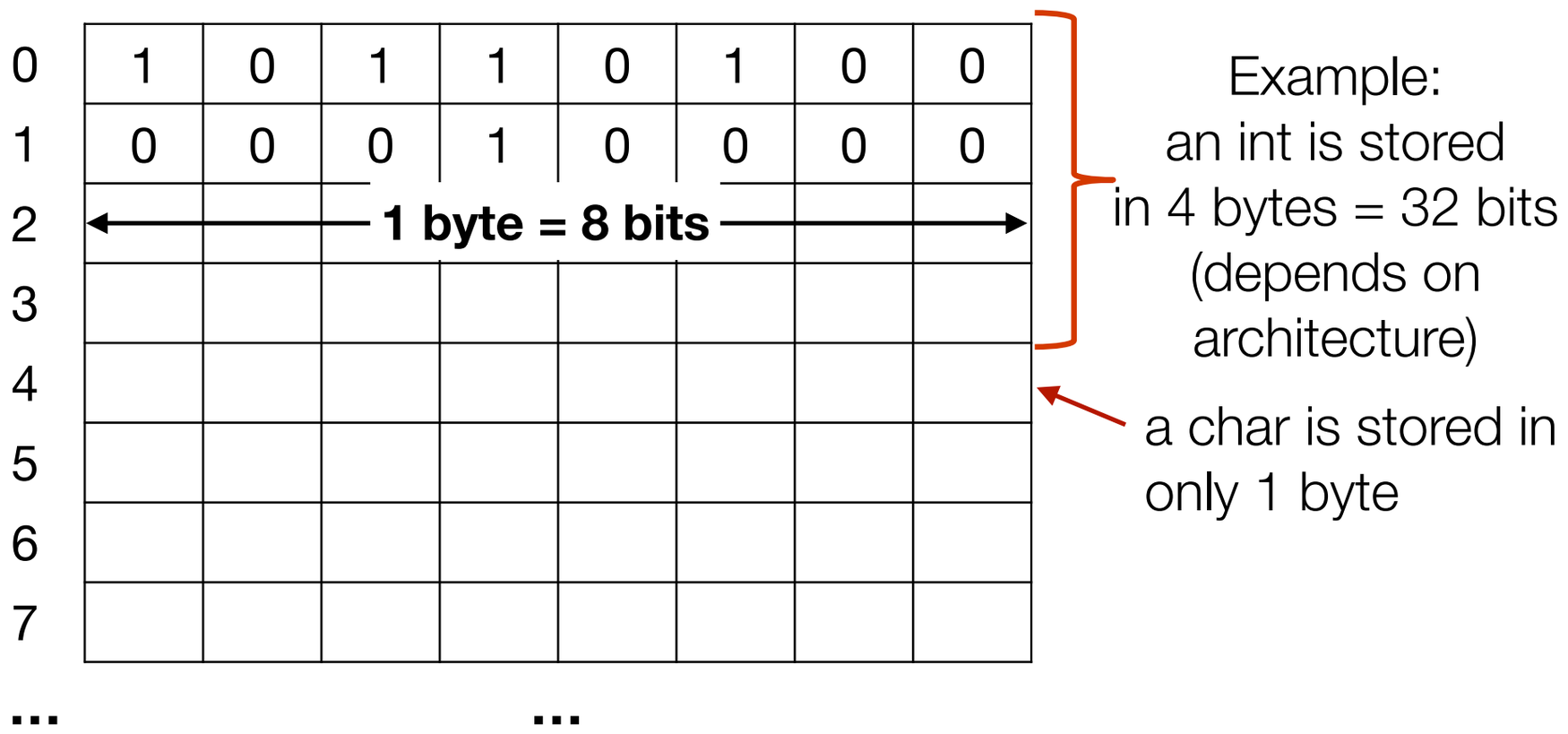
Why are Variable Types Important?

- All variables in your program are stored some place in memory in binary form.



Why are Variable Types Important?

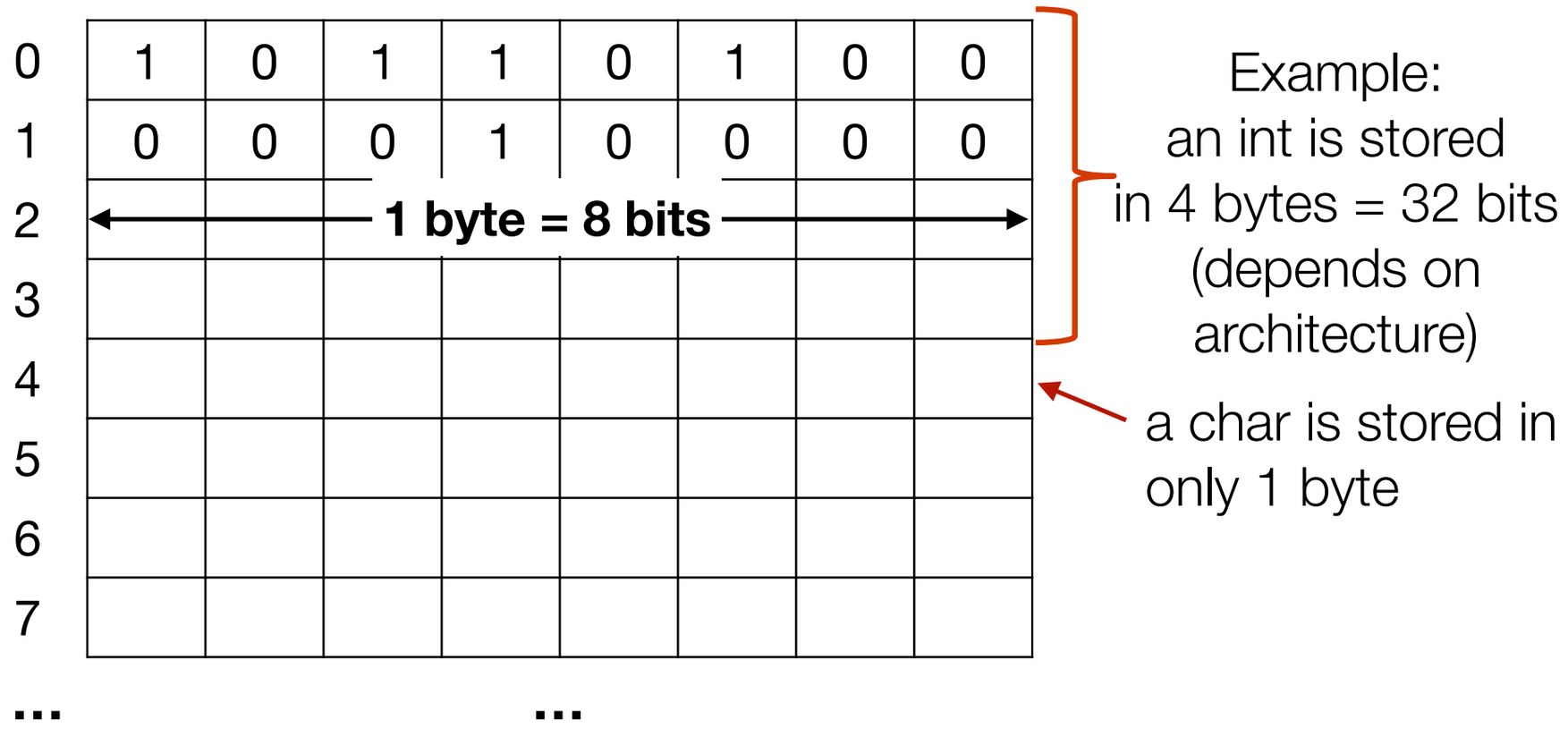
- All variables in your program are stored some place in memory in binary form.



Why are Variable Types Important?

- All variables in your program are stored some place in memory in binary form.

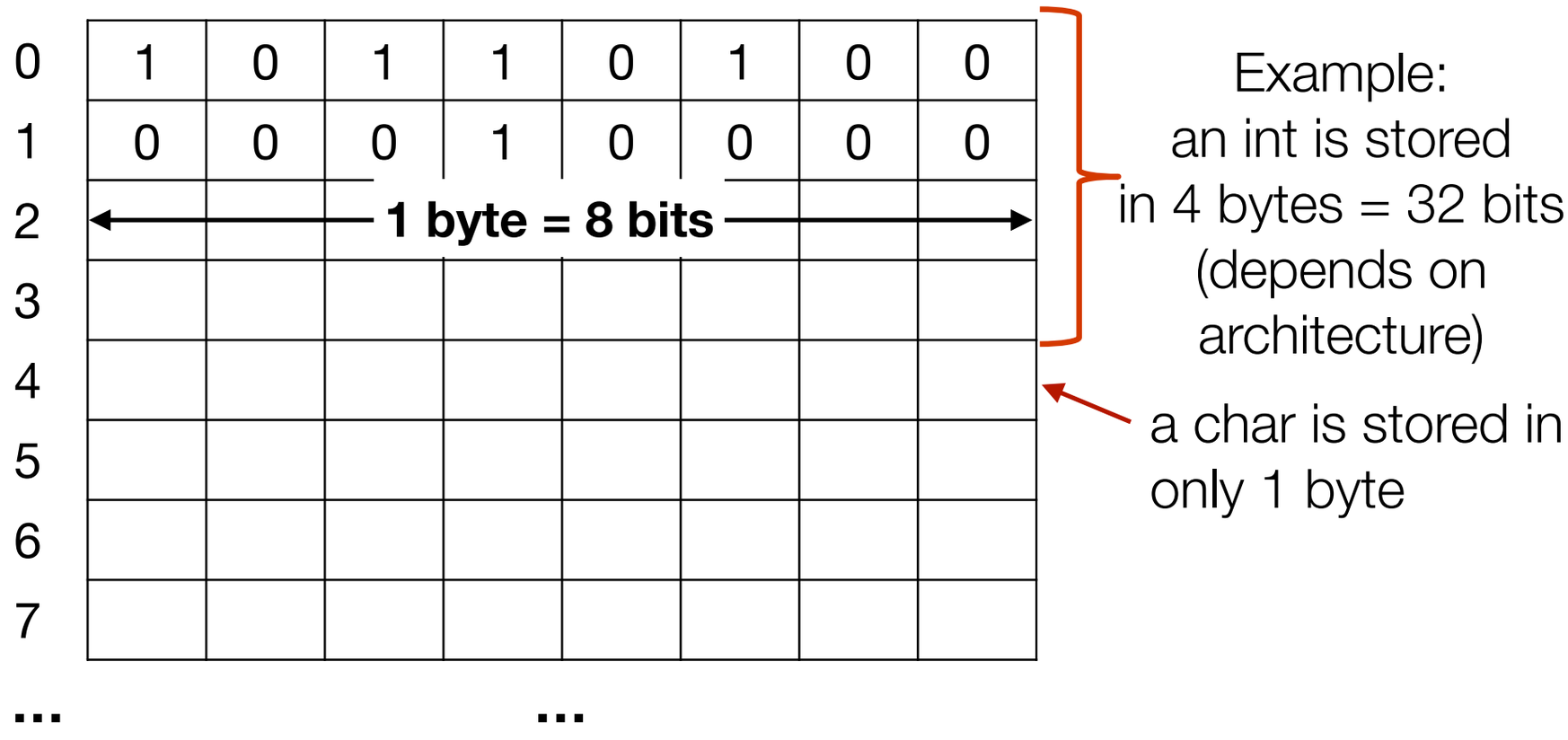
If data occupies a different number of bytes depending on its type, then printf needs to know what variable type it will print such that the compiler knows how much data it needs to read, and how to interpret this data.



Why are Variable Types Important?

- All variables in your program are stored some place in memory in binary form.

If data occupies a different number of bytes depending on its type, then printf needs to know what variable type it will print such that the compiler knows how much data it needs to read, and how to interpret this data.



Basically, printf converts the internal form of data (binary representation) into a printed form (characters to be displayed on the screen)

Computing the Dimensional Weight of a Box — with Input

- Now, assume that instead of hard-coding the dimensions of the box into our program, we want the user to input them.

- We also now want to represent volume and weight as floating point numbers and print them with a precision of 2 places.

- So the program will look like:

```
Enter length of the box: 12
Enter width of the box: 10
Enter height of the box: 8
Volume (cubic inches): 960.00
Dimensional weight (pounds): 6.00
```

Computing the Dimensional Weight of a Box — with Input

- Now, assume that instead of hard-coding the dimensions of the box into our program, we want the user to input them.

- We also now want to represent volume and weight as floating point numbers and print them with a precision of 2 places.

- So the program will look like:

```
Enter length of the box: 12
Enter width of the box: 10
Enter height of the box: 8
Volume (cubic inches): 960.00
Dimensional weight (pounds): 6.00
```

We need a way to read input from the user.

scanf is one of the ways we can do that

The `scanf` Function

- Reads input according to a particular format and stores it into variables: `scanf(string, &var1, &var2, …);`

- The format string follows the same rules as the printf format string, but usually contains only conversion specifications

- `&` precedes each variable (there are exceptions that we will discuss later). Missing this may lead to your program crashing!

- Example:

```
int x;
printf(“Enter value of x:”);
scanf(“%d”, &x); // assume user enters 5
printf(“%d”, x); // this will print 5
```

How `scanf` Works

- Sequentially, for each conversion specification:
  - locate an item of the appropriate type, skipping blank space if necessary
  - if an inappropriate character (can't belong to the item) is encountered:
    - stop processing
    - put this last character back to the input.
    - return the number of values successfully read in
  - If the conversion specification is successfully read, continue processing the format string

Return value of `scanf` example

```
/* Converts a Fahrenheit temperature to Celsius */
#include <stdio.h>
#define FREEZING_PT 32.0f
#define SCALE_FACTOR (5.0f / 9.0f)
int main(void) {
    float fahrenheit, celsius;
    for (;;) {
        printf("Enter Fahrenheit temperature (non-number to quit): ");
        if (scanf("%f", &fahrenheit) == 1) {
            celsius = (fahrenheit - FREEZING_PT) * SCALE_FACTOR;
            printf("Celsius equivalent: %.1f\n", celsius);
        } else
            break;
    }

    return 0;
}
```

demo: celcius_scanf.c

How `scanf` Recognizes Integers & Floating-point Numbers

- For integers:
  - ignoring white space characters in input, search for a plus sign, a minus sign or a digit
  - then continue reading digits until reaching a non-digit
  - Put that non-digit back into the input stream

- For floating-point numbers:
  - ignoring white space characters in input, search for a plus sign, a minus sign, a digit, or a decimal point
  - then continue reading a series of digits (possibly containing a decimal point if it hasn't already encountered one as the first character)
  - lastly, look for a possible exponent (e or E) plus an optional sign (- or +) plus more digits

- For example:

```
int i, j;
float x, y;
scanf("%d%d%f%f", &i, &j, &x, &y);
```

  - input: "1 -20 .3 -4.0e3"
  - input: "1\n -20 \t.3-4.0e3"
  - input: "1-20.3-4.0e3"

How `scanf`'s Format Strings Work:

- Whitespace characters in the format string will match any number of whitespace characters in the input string, including none.

- Other characters in the format string:

  - will be compared to the characters in the input string

    - If equal, scanf will "consume" this character and continue processing. If not, scanf will put the non-matching character back into the input and abort further processing.

    - Example: `scanf("%d/%d", &i, &j);`

      - input: `"5/_9"` (correct: / matches /, space ignored)

      - input: `"5_/9"` (incorrect: space mismatches with /)

Mentimeter `scanf` Question

```
float a, c;
int b;

scanf("%f/%d %f", &a, &b, &c);
```

What will be the values of a, b, and c if the user enters “2.3/3.26_ _20”?
Note that _ is a space

Mentimeter `scanf` Question

```
float a, c;
int b;

scanf("%f/%d %f", &a, &b, &c);
```

What will be the values of a, b, and c if the user enters “2.3/3.26_ _20”?
Note that _ is a space

Quiz using mentimeter

Check out Additional Programs Posted on eClass!

- Compile and run the following programs. Keep an eye out for warnings & errors and then think about the output

  - macro.c — demonstrates the use of macros
  - printf_format.c — see the warning produced?
  - addfrac.c
  - scanf_whitespace.c
  - scanf_matching.c
  - celsius.c

What Exactly is “&”?

- `&x` is the memory address of variable `x`

already has data

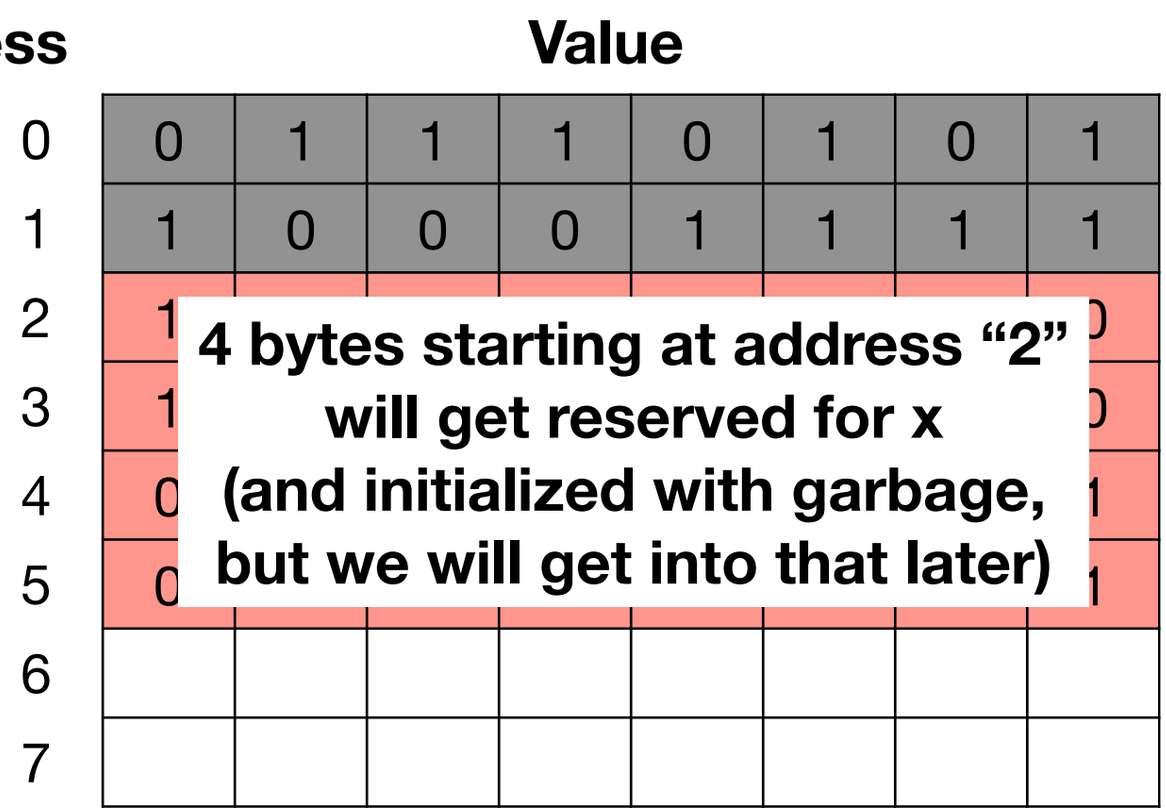
Memory Address	Value							
0	0	1	1	1	0	1	0	1
1	1	0	0	0	1	1	1	1
2								
3								
4								
5								
6								
7								

What Exactly is “&”?

- `&x` is the memory address of variable `x`

already has data

```
int x;
```



What Exactly is “&”?

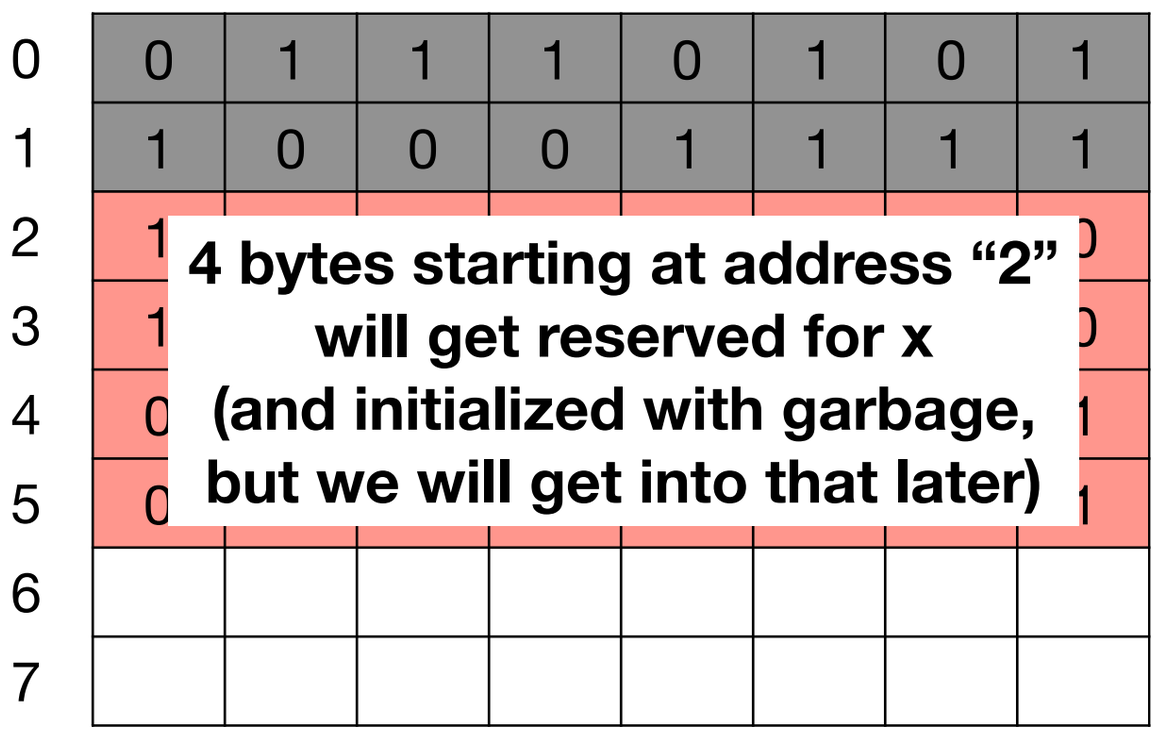
- `&x` is the memory address of variable `x`

already has data

```
int x;

scanf(“%d”, &x);
```

- scanf reads a value from input, translates it into binary form, and then needs to store it somewhere
- So we need to tell scanf where to store this value
- In this particular case, `&x` will have the value of 2 (i.e., address of x)
- This tells scanf that it needs to store the value it read starting at address 2 (number of bytes to be occupied depends on variable type)



What Exactly is “&”?

- `&x` is the memory address of variable `x`

```
int x;

scanf(“%d”, &x);
```

- scanf reads a value from input, translates it into binary form, and then needs to store it somewhere
- So we need to tell scanf where to store this value
- In this particular case, `&x` will have the value of 2 (i.e., address of x)
- This tells scanf that it needs to store the value it read starting at address 2 (number of bytes to be occupied depends on variable type)

already has data

Memory Address	Value							
0	0	1	1	1	0	1	0	1
1	1	0	0	0	1	1	1	1
2	1							0
3	1							0
4	0							1
5	0							1
6								
7								

4 bytes starting at address “2” will get reserved for x (and initialized with garbage, but we will get into that later)

Basically, scanf converts input characters (those entered in terminal) and converts them to internal form of data (binary representation) to be able to store them in memory

Computing the Dimensional Weight of a Box — with Input

- Now, assume that instead of hard-coding the dimensions of the box into our program, we want the user to input them.

- We also now want to represent volume and weight as floating point numbers and print them with a precision of 2 places. The dimensional weight still needs to be the *ceil* of the value. E.g., 1.2 -> 2.00, 1.8 -> 2.00, 2.1 -> 3 etc.

- So the program will look like:

```
Enter length of the box: 12
Enter width of the box: 10
Enter height of the box: 8
Volume (cubic inches): 960.00
Dimensional weight (pounds): 6.00
```

Group Exercise!