CS 31: Intro to Systems C Programming
L03: C programming & Data representation

Vasanta Chaganti & Kevin Webb
Swarthmore College
September 12, 2023
Announcements

• HW1 is due Thursday before class
  • up to groups of four
  • invitations sent from gradescope

• Lab 1 is due Thursday, 11.59 PM

• Clickers will count for credit from this week
Reading Quiz

• Note the red border!

• 1 minute per question

• No talking, no laptops, phones during the quiz
Class today...let’s try something different

• reading quiz (5 mins)
• content block /recap (15 mins)
• group discussions (15 mins)
• content block 2 (10 mins)
• group discussions (15 mins)

---end of class---
Agenda

• C programming
  • arrays, strings
  • functions and stack diagrams
  • structs
  • C is NOT the focus of this course: ask questions if you have them!

• Data representation
  • number systems + conversion
  • data types, storage
  • sizes, representation
  • signedness
Python versus C: Paradigms

https://devrant.com/rants/1755638/c-vs-python
Recap
Recap: Types in C

• All variables have an explicit type!
  – \(<\text{variable type}> \ <\text{variable name}>;\)

• Examples:
  \begin{align*}
  \text{int} & \ \ \text{humidity}; \quad \text{float} \ \ \text{temperature}; \\
  \text{humidity} & \ = \ 20; \quad \text{temperature} \ = \ 32.5
  \end{align*}
Recap: An Example with Local Variables

/* a multiline comment:
   anything between slashdot and dotslash */

#include <stdio.h> // C’s standard I/O library (for printf)

int main() {
  // first: declare main’s local variables
  int x, y;
  float z;

  // followed by: main function statements
  x = 6;
  y = (x + 3)/2; // x and y are both ints
  z = x; // z is a float, value of x is converted to a float
  z = (z + 3)/2;

  printf(…); // Print x, y, z
}

Clicker choices

<table>
<thead>
<tr>
<th></th>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>B</td>
<td>6</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>C</td>
<td>6</td>
<td>4.5</td>
<td>4</td>
</tr>
<tr>
<td>D</td>
<td>6</td>
<td>4</td>
<td>4.5</td>
</tr>
<tr>
<td>E</td>
<td>6</td>
<td>4.5</td>
<td>4.5</td>
</tr>
</tbody>
</table>
Recap: Boolean values in C

- Zero (0) is **false**, any non-zero value is **true**

- **Logical** (operands int “boolean”->result int “boolean”):
  - ! (not): inverts truth value
  - && (and): true if both operands are true
  - || (or): true if either operand is true

Do the following statements evaluate to True or False?

#1:  
\((!10) \text{ || } (5 > 2)\)

#2:  
\((-1) \text{ && } ((!5) > -1)\)

<table>
<thead>
<tr>
<th></th>
<th>#1</th>
<th>#2</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>B</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>C</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>D</td>
<td>False</td>
<td>False</td>
</tr>
</tbody>
</table>
Recap: Conditional Statements

<table>
<thead>
<tr>
<th>Chaining if-else if</th>
<th>With optional else:</th>
</tr>
</thead>
<tbody>
<tr>
<td>if(&lt;boolean expr1&gt;) {</td>
<td>if(&lt;boolean expr1&gt;) {</td>
</tr>
<tr>
<td>if-expr1-true-body</td>
<td>if-expr1-true-body</td>
</tr>
<tr>
<td>} else if (&lt;bool expr2&gt;){</td>
<td></td>
</tr>
<tr>
<td>else-if-expr2-true-body</td>
<td>else-if-expr2-true-body</td>
</tr>
<tr>
<td>(expr1 false)</td>
<td>}</td>
</tr>
<tr>
<td>}</td>
<td></td>
</tr>
</tbody>
</table>
| ... | ...
| } else if (<bool exprN>){
|     else-if-exprN-true-body | } else if (<bool exprN>){
| } |     else-if-exprN-true-body |
| } else if (<bool exprN>){
|     else-if-exprN-true-body | } else {
| } |     else body |
| } | (all exprX’s false) |
| } | }

Very similar to Python, just remember {} are blocks
Recap: For loops: different than Python’s

```
for (<init>; <cond>; <step>) {
    for-loop-body-statements
}

<next stmt after loop>;
```

1. **Evaluate <init> one time**, when first eval for statement
2. Evaluate <cond>, if it is false, drop out of the loop (<next stmt after>)
3. Evaluate the statements in the for loop body
4. Evaluate <step>
5. **Goto step (2)**

```c
for(i=1; i <= 10; i++) {  // example for loop
    printf("%d\n", i*i);
}
```

What does this for loop print?
Recap: While Loops

Basically identical to Python while loops:

```c
while(<boolean expr>) {
   while-expr-true-body
}
```

```
x = 20;
while (x < 100) {
    y = y + x;
    x += 4;  // x = x + 4;
} <next stmt after loop>;
```

```
x = 20;
while(1) {
    y = y + x;
    x += 4;
    if(x >= 100) {
        break;  // break out of loop
    }
} <next stmt after loop>;
```
Data Collections in C

• Many complex data types out there (CS 35)

• C has a few simple ones built-in:
  – Arrays
  – Strings (arrays of characters)
  – Structures (struct)

• Often combined in practice, e.g.:
  – An array of structs
  – A struct containing strings
Arrays and Strings

- C’s support for **collections of values**
  - Array buckets store a single type of value
  - There is no “string” data type 😞
  - **Specify max capacity** (num buckets) when you declare an array variable (single memory chunk)
    
    ```c
    <type> <var_name>[<num_buckets>];
    
    int arr[5];  // an array of 5 integers
    float rates[40]; // an array of 40 floats
    ```
Array Characteristics

```c
int january_temps[31]; // Daily high temps
```

• Indices start at 0! Why?
• Array variable name means, to the compiler, the beginning of the memory chunk. (The memory address)
  – `january_temps` (without brackets!) **Location of [0] in memory.**
  – Keep this in mind, we’ll return to it soon (functions).
Array Characteristics

```
int january_temps[31];  // Daily high temps
```

- Indices start at 0! Why?
- The index refers to an offset from the start of the array
  - e.g., `january_temps[3]` means “three integers forward from the starting address of `january_temps`”
Characters and Strings

A character (type \texttt{char}) is numerical value that holds one letter.
\begin{verbatim}
char my_letter = 'w'; // Note: single quotes
\end{verbatim}

What is the numerical value?
\begin{itemize}
  \item \texttt{printf("\%d \%c", my_letter, my_letter)};
  \item Would print: 119 w
\end{itemize}

Why is ‘w’ equal to 119?
\begin{itemize}
  \item ASCII Standard says so.
  \item American Standard Code for Information Interchange
\end{itemize}
<table>
<thead>
<tr>
<th>Dec</th>
<th>Hex</th>
<th>Char</th>
<th>Dec</th>
<th>Hex</th>
<th>Char</th>
<th>Dec</th>
<th>Hex</th>
<th>Char</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00</td>
<td>Null</td>
<td>32</td>
<td>20</td>
<td>Space</td>
<td>64</td>
<td>40</td>
<td>@</td>
</tr>
<tr>
<td>1</td>
<td>01</td>
<td>Start of heading</td>
<td>33</td>
<td>21</td>
<td>!</td>
<td>65</td>
<td>41</td>
<td>A</td>
</tr>
<tr>
<td>2</td>
<td>02</td>
<td>Start of text</td>
<td>34</td>
<td>22</td>
<td>&quot;</td>
<td>66</td>
<td>42</td>
<td>B</td>
</tr>
<tr>
<td>3</td>
<td>03</td>
<td>End of text</td>
<td>35</td>
<td>23</td>
<td>#</td>
<td>67</td>
<td>43</td>
<td>C</td>
</tr>
<tr>
<td>4</td>
<td>04</td>
<td>End of transmit</td>
<td>36</td>
<td>24</td>
<td>$</td>
<td>68</td>
<td>44</td>
<td>D</td>
</tr>
<tr>
<td>5</td>
<td>05</td>
<td>Enquiry</td>
<td>37</td>
<td>25</td>
<td>%</td>
<td>69</td>
<td>45</td>
<td>E</td>
</tr>
<tr>
<td>6</td>
<td>06</td>
<td>Acknowledge</td>
<td>38</td>
<td>26</td>
<td>&amp;</td>
<td>70</td>
<td>46</td>
<td>F</td>
</tr>
<tr>
<td>7</td>
<td>07</td>
<td>Audible bell</td>
<td>39</td>
<td>27</td>
<td>'</td>
<td>71</td>
<td>47</td>
<td>G</td>
</tr>
<tr>
<td>8</td>
<td>08</td>
<td>Backspace</td>
<td>40</td>
<td>28</td>
<td>(</td>
<td>72</td>
<td>48</td>
<td>H</td>
</tr>
<tr>
<td>9</td>
<td>09</td>
<td>Horizontal tab</td>
<td>41</td>
<td>29</td>
<td>)</td>
<td>73</td>
<td>49</td>
<td>I</td>
</tr>
<tr>
<td>10</td>
<td>0A</td>
<td>Line feed</td>
<td>42</td>
<td>2A</td>
<td>^</td>
<td>74</td>
<td>4A</td>
<td>J</td>
</tr>
<tr>
<td>11</td>
<td>0B</td>
<td>Vertical tab</td>
<td>43</td>
<td>2B</td>
<td>+</td>
<td>75</td>
<td>4B</td>
<td>K</td>
</tr>
<tr>
<td>12</td>
<td>0C</td>
<td>Form feed</td>
<td>44</td>
<td>2C</td>
<td>,</td>
<td>76</td>
<td>4C</td>
<td>L</td>
</tr>
<tr>
<td>13</td>
<td>0D</td>
<td>Carriage return</td>
<td>45</td>
<td>2D</td>
<td>-</td>
<td>77</td>
<td>4D</td>
<td>M</td>
</tr>
<tr>
<td>14</td>
<td>0E</td>
<td>Shift out</td>
<td>46</td>
<td>2E</td>
<td>.</td>
<td>78</td>
<td>4E</td>
<td>N</td>
</tr>
<tr>
<td>15</td>
<td>0F</td>
<td>Shift in</td>
<td>47</td>
<td>2F</td>
<td>/</td>
<td>79</td>
<td>4F</td>
<td>O</td>
</tr>
<tr>
<td>16</td>
<td>10</td>
<td>Data link escape</td>
<td>48</td>
<td>30</td>
<td>0</td>
<td>80</td>
<td>50</td>
<td>P</td>
</tr>
<tr>
<td>17</td>
<td>11</td>
<td>Device control 1</td>
<td>49</td>
<td>31</td>
<td>1</td>
<td>81</td>
<td>51</td>
<td>Q</td>
</tr>
<tr>
<td>18</td>
<td>12</td>
<td>Device control 2</td>
<td>50</td>
<td>32</td>
<td>2</td>
<td>82</td>
<td>52</td>
<td>R</td>
</tr>
<tr>
<td>19</td>
<td>13</td>
<td>Device control 3</td>
<td>51</td>
<td>33</td>
<td>3</td>
<td>83</td>
<td>53</td>
<td>S</td>
</tr>
<tr>
<td>20</td>
<td>14</td>
<td>Device control 4</td>
<td>52</td>
<td>34</td>
<td>4</td>
<td>84</td>
<td>54</td>
<td>T</td>
</tr>
<tr>
<td>21</td>
<td>15</td>
<td>Neg. acknowledge</td>
<td>53</td>
<td>35</td>
<td>5</td>
<td>85</td>
<td>55</td>
<td>U</td>
</tr>
<tr>
<td>22</td>
<td>16</td>
<td>Synchronous idle</td>
<td>54</td>
<td>36</td>
<td>6</td>
<td>86</td>
<td>56</td>
<td>V</td>
</tr>
<tr>
<td>23</td>
<td>17</td>
<td>End trans. block</td>
<td>55</td>
<td>37</td>
<td>7</td>
<td>87</td>
<td>57</td>
<td>W</td>
</tr>
<tr>
<td>24</td>
<td>18</td>
<td>Cancel</td>
<td>56</td>
<td>38</td>
<td>0</td>
<td>88</td>
<td>58</td>
<td>X</td>
</tr>
<tr>
<td>25</td>
<td>19</td>
<td>End of medium</td>
<td>57</td>
<td>39</td>
<td>9</td>
<td>89</td>
<td>59</td>
<td>Y</td>
</tr>
<tr>
<td>26</td>
<td>1A</td>
<td>Substitution</td>
<td>58</td>
<td>3A</td>
<td>:</td>
<td>90</td>
<td>5A</td>
<td>Z</td>
</tr>
<tr>
<td>27</td>
<td>1B</td>
<td>Escape</td>
<td>59</td>
<td>3B</td>
<td>;</td>
<td>91</td>
<td>5B</td>
<td>[</td>
</tr>
<tr>
<td>28</td>
<td>1C</td>
<td>File separator</td>
<td>60</td>
<td>3C</td>
<td>&lt;</td>
<td>92</td>
<td>5C</td>
<td>\</td>
</tr>
<tr>
<td>29</td>
<td>1D</td>
<td>Group separator</td>
<td>61</td>
<td>3D</td>
<td>-</td>
<td>93</td>
<td>5D</td>
<td>]</td>
</tr>
<tr>
<td>30</td>
<td>1E</td>
<td>Record separator</td>
<td>62</td>
<td>3E</td>
<td>&gt;</td>
<td>94</td>
<td>5E</td>
<td>^</td>
</tr>
<tr>
<td>31</td>
<td>1F</td>
<td>Unit separator</td>
<td>63</td>
<td>3F</td>
<td>?</td>
<td>95</td>
<td>5F</td>
<td>\</td>
</tr>
</tbody>
</table>
Characters and Strings

• A character (type `char`) is a numerical value that holds one letter.
• A string is a memory block containing characters, one after another...

Hmm, suppose we used printf and %s to print name.

• Examples:

```plaintext
char food[6] = "Pizza";
```

How does it know where the string ends and other memory begins?

```
  P i z z a
  [0] [1] [2] [3] [4]
```
0 is the “Null character”

Special stuff over here in the lower values.

Characters and Strings

$ man ascii
Characters and Strings

• A character (type `char`) is numerical value that holds one letter.
• A string is a memory block containing characters, one after another, with a null terminator (numerical 0) at the end.
• Examples:
  ```
  char name[20] = “Pizza”;
  ```
Strings in C

• C String library functions: `#include <string.h>`
  – Common functions (strlen, strcpy, etc.) make strings easier
  – Less friendly than Python strings

• More on strings later, in labs.

• For now, remember about strings:
  – Allocate enough space for null terminator!
  – If you’re modifying a character array (string), don’t forget to set the null terminator!
  – If you see crazy, unpredictable behavior with strings, check these two things!
Functions and Stack Diagrams
Functions: Specifying Types

Need to specify the return type of the function, and the type of each parameter:

```c
<return type> <func name> ( <param list> ) {
    // declare local variables first
    // then function statements
    return <expression>;
}

// my_function takes 2 int values and returns an int
int my_function(int x, int y) {
    int result;
    result = x;
    if(y > x) {
        result = y+5;
    }
    return result*2;
}
```

Compiler will yell at you if you try to pass the wrong type!
Arguments are **passed by value**

- The function gets a separate copy of the passed variable

```c
int func(int a, int b) {
    a = a + 5;
    return a - b;
}
```

```c
int main() {
    // declare two integers
    int x, y;
    x = 4;
    y = 7;
    y = func(x, y);
    printf("%d, %d", x, y);
}
```
Arguments are **passed by value**

- The function gets a separate **copy** of the passed variable

```c
int func(int a, int b) {
    a = a + 5;
    return a - b;
}

int main() {
    // declare two integers
    int x, y;
    x = 4;
    y = 7;
    y = func(x, y);
    printf("%d, %d", x, y);
}
```

```
Stack

main:

x: 4

y: 7
```
Arguments are passed by value
– The function gets a separate copy of the passed variable

```c
int func(int a, int b) {
    a = a + 5;
    return a - b;
}

int main() {
    // declare two integers
    int x, y;
    x = 4;
    y = 7;
    y = func(x, y);
    printf("%d, %d", x, y);
}
```
Arguments are **passed by value**

– The function gets a separate copy of the passed variable

```c
int func(int a, int b) {
    a = a + 5;
    return a - b;
}

int main() {
    // declare two integers
    int x, y;
    x = 4;
    y = 7;
    y = func(x, y);
    printf("%d, %d", x, y);
}
```
Function Arguments

Arguments are **passed by value**

— The function gets a separate **copy** of the passed variable

```c
int func(int a, int b) {
    a = a + 5;
    return a - b;
}

int main() {
    // declare two integers
    int x, y;
    x = 4;
    y = 7;
    y = func(x, y);
    printf("%d, %d", x, y);
}
```

No impact on values in main!
Function Arguments

Arguments are **passed by value**

– The function gets a separate **copy** of the passed variable

```c
int func(int a, int b) {
    a = a + 5;
    return a - b;
}
```

```c
int main() {
    // declare two integers
    int x, y;
    x = 4;
    y = 7;
    y = func(x, y);
    printf("%d, %d", x, y);
}
```

Stack

`main:
 x: 4
 y: 2`
Function Arguments

Arguments are **passed by value**
- The function gets a separate **copy** of the passed variable

```c
int func(int a, int b) {
    a = a + 5;
    return a - b;
}

int main() {
    // declare two integers
    int x, y;
    x = 4;
    y = 7;
    y = func(x, y);
    printf("%d, %d", x, y);
}
```

**Output:** 4, 2
```c
int func(int a, int y, int my_array[]) {
  y = 1;
  my_array[a] = 0;
  my_array[y] = 8;
  return y;
}

int main() {
  int x;
  int values[2];

  x = 0;
  values[0] = 5;
  values[1] = 10;

  x = func(x, x, values);

  printf("%d, %d, %d", x, values[0], values[1]);
}
```

What will this print?

A. 0, 5, 8
B. 0, 5, 10
C. 1, 0, 8
D. 1, 5, 8
E. 1, 5, 10

Hint: What does the name of an array mean to the compiler?
```c
int func(int a, int y, int my_array[]) {
    y = 1;
    my_array[a] = 0;
    my_array[y] = 8;
    return y;
}

int main() {
    int x;
    int values[2];

    x = 0;
    values[0] = 5;
    values[1] = 10;

    x = func(x, x, values);

    printf("%d, %d, %d", x, values[0], values[1]);
}
```

What will this print?

A. 0, 5, 8  
B. 0, 5, 10  
C. 1, 0, 8  
D. 1, 5, 8  
E. 1, 5, 10

Hint: Still accessing the same memory location of array in `func`
Discussion Block 1
structs

• Treat a collection of values as a single type:
  – C is not an object oriented language, no classes
  – A struct is similar to the data part of a class

• Rules:
  1. Define a new struct type outside of any function
  2. Declare variables of the new struct type
  3. Use dot notation to access the field values of a struct variable
Struct Example

Suppose we want to represent a student type.

```c
struct student {
    char name[20];
    int grad_year;
    float gpa;
};
// Variable bob is of type struct student
struct student bob;
// Set name (string) with strcpy()
strcpy(bob.name, "Robert Paulson");
bob.grad_year = 2019;
bob.gpa = 3.1;

printf("Name: %s, year: %d, GPA: %f", bob.name, bob.grad_year, bob.gpa);
```
Arrays of Structs

```c
struct student {
    char name[20];
    int grad_year;
    float gpa;
};

//create an array of struct students!
struct student classroom[50];

strcpy(classroom[0].name, "Alice");
classroom[0].grad_year = 2023
classroom[0].gpa = 4.0;

// With a loop, create an army of Alice clones!
int i;
for (i = 0; i < 50; i++) {
    strcpy(classroom[i].name, "Alice");
classroom[i].grad_year = 2023;
classroom[i].gpa = 4.0;
}
```
Arrays of Structs

```c
struct student classroom[3];

strcpy(classroom[0].name, "Alice");
classroom[0].grad_year = 2021;
classroom[0].gpa = 4.0;

strcpy(classroom[1].name, "Bob");
classroom[1].grad_year = 2022;
classroom[1].gpa = 3.1

strcpy(classroom[2].name, "Cat");
classroom[2].grad_year = 2023;
classroom[2].gpa = 3.4
```
Array of Structs: Layout in Memory

classroom: array of structs

<table>
<thead>
<tr>
<th>‘A’</th>
<th>‘l’</th>
<th>‘i’</th>
<th>‘c’</th>
<th>‘e’</th>
<th>‘\0’</th>
<th>...</th>
<th>‘B’</th>
<th>‘o’</th>
<th>‘b’</th>
<th>‘\0’</th>
<th>...</th>
<th>‘C’</th>
<th>‘a’</th>
<th>‘t’</th>
<th>‘\0’</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>2021</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2022</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2023</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Abstraction

User / Programmer
Wants low complexity

Applications
Specific functionality

Software library
Reusable functionality

Operating system
Manage resources

Complex devices
Compute & I/O
Data Storage

• Lots of technologies out there:
  – Magnetic (hard drive, floppy disk)
  – Optical (CD / DVD / Blu-Ray)
  – Electronic (RAM, registers, …)

• Focus on electronic for now
  – We’ll see (and build) digital circuits soon

• Relatively easy to differentiate two states
  – Voltage present
  – Voltage absent
Bits and Bytes

• **Bit**: a 0 or 1 value (binary)
  – HW represents as two different voltages
    • 1: the presence of voltage (*high voltage*)
    • 0: the absence of voltage (*low voltage*)

• **Byte**: 8 bits, the *smallest addressable unit*
  
  Memory: 01010101 10101010 00001111 ...
  (address) [0] [1] [2] ...

• Other names:
  – 4 bits: Nibble
  – “Word”: Depends on system, often 4 bytes
Files

Sequence of bytes... nothing more, nothing less
Binary Digits (BITs)

- One bit: two values (0 or 1)
- Two bits: four values (00, 01, 10, or 11)
- Three bits: eight values (000, 001, ..., 110, 111)
How many unique values can we represent with 9 bits? Why?

- One bit: two values (0 or 1)
- Two bits: four values (00, 01, 10, or 11)
- Three bits: eight values (000, 001, ..., 110, 111)

A. 18
B. 81
C. 256
D. 512
E. Some other number of values.
How many unique values can we represent with 9 bits? Why?

- One bit: two values (0 or 1)
- Two bits: four values (00, 01, 10, or 11)
- Three bits: eight values (000, 001, ..., 110, 111)

A. 18
B. 81
C. 256
D. 512
E. Some other number of values.
How many values?

1 bit: 0 1
How many values?

1 bit:

2 bits:
How many values?

1 bit: 0, 1

2 bits: 0 0, 0 1, 1 0, 1 1

3 bits: 0 0 0, 0 0 1, 0 1 0, 0 1 1, 1 0 0, 1 0 1, 1 1 0, 1 1 1
How many values?

1 bit:

2 bits:

3 bits:

4 bits:

N bits: $2^N$ values
C types and their (typical!) sizes

• 1 byte: char, unsigned char
• 2 bytes: short, unsigned short
• 4 bytes: int, unsigned int, float
• 8 bytes: long long, unsigned long long, double
• 4 or 8 bytes: long, unsigned long

unsigned long v1;
short s1;
long long ll;

// prints out number of bytes
printf("%lu %lu %lu\n", sizeof(v1), sizeof(s1), sizeof(ll));

WARNING: These sizes are **NOT** a guarantee. Don't always assume that every system will use these values!

How do we use this storage space (bits) to represent a value?
Let’s start with what we know…

• Digits 0-9

• **Positional numbering**

• Digits are composed to make larger numbers

• Known as **Base 10** representation
Decimal number system (Base 10)

• Sequence of digits in range [0, 9]

64025

Digit #0: 1’s place, “least significant digit”

Digit #1: 10’s place

Digit #4: “most significant digit”
Decimal: Base 10

A number, written as the sequence of N digits,

\[ d_{n-1} \ldots d_2 d_1 d_0 \]

where \(d\) is in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}, represents the value:

\[ [d_{n-1} \times 10^{n-1}] + [d_{n-2} \times 10^{n-2}] + \ldots + [d_1 \times 10^1] + [d_0 \times 10^0] \]

64025 =

\[ 6 \times 10^4 \quad + \quad 4 \times 10^3 \quad + \quad 0 \times 10^2 \quad + \quad 2 \times 10^1 \quad + \quad 5 \times 10^0 \]

60000 + 4000 + 0 + 20 + 5
Binary: Base 2

• Used by computers to store digital values.

• **Indicated by prefixing number with** $0b$

• A number, written as the sequence of $N$ digits, $d_{n-1}...d_2d_1d_0$, where $d$ is in \{0,1\}, represents the value:

$$[d_{n-1} \times 2^{n-1}] + [d_{n-2} \times 2^{n-2}] + ... + [d_2 \times 2^2] + [d_1 \times 2^1] + [d_0 \times 2^0]$$
Converting Binary to Decimal

Most significant bit $\underline{10001111}$ Least significant bit

representation:

$1 \times 2^7 + 0 \times 2^6 \ldots + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$

$128 + 8 + 4 + 2 + 1$

$10001111 = 143$
Hexadecimal: Base 16

- **Indicated by prefixing number with 0x**

A number, written as the sequence of N digits,

\[ d_{n-1}...d_2d_1d_0, \]

where \( d \) is in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}, represents:

\[
[d_{n-1} \times 16^{n-1}] + [d_{n-2} \times 16^{n-2}] + ... + [d_2 \times 16^2] + [d_1 \times 16^1] + [d_0 \times 16^0]
\]
Generalizing: Base $b$

- The meaning of a digit depends on its position in a number.

A number, written as the sequence of $N$ digits,

$$d_{n-1} \ldots d_2 d_1 d_0$$

in base $b$ represents the value:

$$[d_{n-1} \times b^{n-1}] + [d_{n-2} \times b^{n-2}] + \ldots + [d_2 \times b^2] + [d_1 \times b^1] + [d_0 \times b^0]$$

Base 10: $$[d_{n-1} \times 10^{n-1}] + [d_{n-2} \times 10^{n-2}] + \ldots + [d_1 \times 10^1] + [d_0 \times 10^0]$$
Other (common) number systems.

- Base 2: How data is stored in hardware.
- Base 8: Used to represent file permissions.
- Base 10: Preferred by people.
- Base 16: Convenient for representing memory addresses.
- Base 64: Commonly used on the Internet, (e.g. email attachments).

It’s all stored as binary in the computer.

Different representations (or visualizations) of the same information!
Discussion block 2
Important Point…

• You can represent the same value in a variety of number systems or bases.

• It’s **all** stored as binary in the computer.
  – Presence/absence of voltage.
Hexadecimal: Base 16

• Fewer digits to represent same value
  – Same amount of information!

• Like binary, the base is power of 2

• Each digit is a “nibble”, or half a byte.
Each hex digit is a “nibble”

- One hex digit: 16 possible values (0-9, A-F)

- $16 = 2^4$, so each hex digit has exactly four bits worth of information.

- We can map each hex digit to a four-bit binary value. (helps for converting between bases)
Each hex digit is a “nibble”

Example value: 0x1B7

Four-bit value: 1
Four-bit value: B (decimal 11)
Four-bit value: 7

In binary: 0001 1011 0111
1   B   7
Converting Decimal -> Binary

• Two methods:
  – division by two remainder
  – powers of two and subtraction
Method 1: decimal value $D$, binary result $b$ ($b_i$ is $i$th digit):

\[ i = 0 \]
\[ \text{while } (D > 0) \]
\[ \quad \text{if } D \text{ is odd} \]
\[ \quad \quad \text{set } b_i \text{ to 1} \]
\[ \quad \text{if } D \text{ is even} \]
\[ \quad \quad \text{set } b_i \text{ to 0} \]
\[ i++ \]
\[ D = D/2 \]

idea: example: $D = 105 \quad b_0 = 1$
Method 1: decimal value D, binary result b (b_i is ith digit):

\[
i = 0 \\
\text{while (D > 0)} \\
\quad \text{if D is odd} \\
\quad \quad \text{set } b_i \text{ to 1} \\
\quad \text{if D is even} \\
\quad \quad \text{set } b_i \text{ to 0} \\
\quad i++ \\
D = D/2
\]

idea: \( D \)  
example: \( D = 105 \)  \( b_0 = 1 \)  
\( D = D/2 \)  \( D = 52 \)  \( b_1 = 0 \)
Method 1: decimal value D, binary result b ($b_i$ is $i$th digit):

$$i = 0$$

while ($D > 0$)

  if $D$ is odd
    set $b_i$ to 1
  else if $D$ is even
    set $b_i$ to 0

  $i$++
  $D = D/2$

idea: $D$ example: $D = 105$

$D = D/2$ example: $D = 52$
$D = D/2$ example: $D = 26$
$D = D/2$ example: $D = 13$
$D = D/2$ example: $D = 6$
$D = D/2$ example: $D = 3$
$D = D/2$ example: $D = 1$
$D = 0$ (done) example: $D = 0$

$$105 = 01101001$$
Method 2

- $2^0 = 1$,  $2^1 = 2$,  $2^2 = 4$,  $2^3 = 8$,  $2^4 = 16$,  $2^5 = 32$,  $2^6 = 64$,  $2^7 = 128$

To convert 105:
- Find largest power of two that’s less than 105 (64)
- Subtract 64 (105 – 64 = 41), put a 1 in $d_6$
- Subtract 32 (41 – 32 = 9), put a 1 in $d_5$
- Skip 16, it’s larger than 9, put a 0 in $d_4$
- Subtract 8 (9 – 8 = 1), put a 1 in $d_3$
- Skip 4 and 2, put a 0 in $d_2$ and $d_1$
- Subtract 1 (1 – 1 = 0), put a 1 in $d_0$ (Done)

\[
\begin{array}{cccccccc}
 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\
\hline \\
\frac{1}{d_6} & \frac{1}{d_5} & \frac{0}{d_4} & \frac{1}{d_3} & \frac{0}{d_2} & \frac{0}{d_1} & \frac{1}{d_0} \\
\end{array}
\]
What is the value of 357 in binary?

A. 101100011
B. 101100101
C. 101101001
D. 101110101
E. 110100101

\[
2^0 = 1, \quad 2^1 = 2, \quad 2^2 = 4, \quad 2^3 = 8, \quad 2^4 = 16, \\
2^5 = 32, \quad 2^6 = 64, \quad 2^7 = 128, \quad 2^8 = 256
\]