Today's Goals

- Number Systems
- Pointers
  - Declaration
  - Assignment
  - Indirection/de-referencing

Section 1

Decimal Number System

(Base-10 number system)

123
$= \mathbf{1} \times 100 + \mathbf{2} \times 10 + \mathbf{3}$
$= \mathbf{1} \times 10^2 + \mathbf{2} \times 10^1 + \mathbf{3} \times 10^0$
$= 123_{10}$

base

Binary Number System

(Base-2 number system)

Bit: **B**inary Digi**t**

101_2
$= \mathbf{1} \times 2^2 + \mathbf{0} \times 2^1 + \mathbf{1} \times 2^0$
$= \mathbf{1} \times 4 + \mathbf{0} \times 2 + \mathbf{1}$
$= 5_{10}$

Capacity of Binary Numbers

- 1 bit can distinguish 2 states (0 or 1).
- An *n*-bit binary number can distinguish 2^n states.

Octal Number System

(Base-8 number system)

173_8
$= \mathbf{1} \times 8^2 + \mathbf{7} \times 8^1 + \mathbf{3} \times 8^0$
$= \mathbf{1} \times 64 + \mathbf{7} \times 8 + \mathbf{3}$
$= 123_{10}$

Hexadecimal Number System

(Base-16 number system)

Character correspondence:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

$9AB_{16}$
$= \mathbf{9} \times 16^2 + \mathbf{10} \times 16^1 + \mathbf{11} \times 16^0$
$= \mathbf{9} \times 256 + \mathbf{10} \times 16 + \mathbf{11}$
$= 2475_{10}$

ex

Byte

1 Byte

- 10001101_2
- 8 bits – can distinguish 256 (2^8) states
- Representable by 2 hexadecimal characters
- $\underbrace{1000}_{8_{16}}\underbrace{1101}_{D_{16}}{}_2$

Kilobyte – KB

- Commonly denoted as KB, Kb, Kbyte or just K.
- Equal to 1000 or 1024 (2^{10}) bytes, depending on whom you ask.
- One Kb then can distinguish

$$2^{10} \times 2^8 = 2^{18} = 262{,}144$$

Section 2

Base-*k*-to-Decimal Conversion

- 101_2
  $= \mathbf{1} \times 2^2 + \mathbf{0} \times 2^1 + \mathbf{1} \times 2^0$
  $= \mathbf{1} \times 4 + \mathbf{0} \times 2 + \mathbf{1}$
  $= 5_{10}$

More generally

$$value_{10} = \sum_{i=0}^{n} (p_i \times k^i)$$

- ☺☯💣$_k$ =
  $= ☺ \times k^2 + ☯ \times k^1 + 💣 \times k^0$
  $= ?_{10}$

Decimal-to-Binary Conversion

$5_{10} = 101_2$

2) 5
2) 2 ... 1 ← the rightmost bit (LSB)
1 ... 0 ← the second bit from the right
↑
the leftmost bit (MSB)

MSB = Most Significant Bit
LSB = Least Significant Bit

Decimal to hexadecimal

$2475_{10} = 9AB_{16}$

16) 2475
16) 154 ... 11 ← the rightmost bit (LSB)
9 ... 10 ←second bit from the right
↑
the leftmost bit (MSB)

Decimal to Base-*k* conversions work the same way

Section 3

Common C Data Types

Type	Size		Largest value	Smallest value
	[bit]	[byte]		
`int`	32	4	2×10^9	-2×10^9
`float`	32	4	10^{38}	-10^{38}
`double`	64	8	10^{308}	-10^{308}
`char`	8	1	127	-128

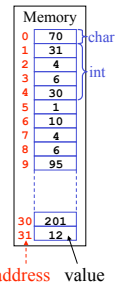
`Double` stands for "double-precision floating point".

- Based on 32-bit architecture
- Shaded values are approximate.
- Precision of `float` is 6 digits, `double` is 9-15 digits.

Section 4

Variable and Address

- Variable = Storage in computer memory
  - Contains some value
  - Must reside at a specific location called *address*
  - Basic unit – byte
  - Imagine memory as a one-dimensional array with addresses as byte indices
  - A variable consists of one or more bytes, depending on its type (size)



Pointer – Reference

- A pointer (pointer variable) is a variable that stores an address (like Java reference)
  - value – address of some memory
  - type – size of that memory
- Recall in Java, when one declares variables of a *class* type, these are automatically references.
- In C, pointers have special syntax and much greater flexibility.

Memory and Address

- A machine with 16 Megabytes of memory has ? bytes $16 \times 2^{20} = 2^4 \times 2^{20} = 16{,}777{,}216$
- Since each byte has a unique address, there are at least that many addresses
- A pointer stores a memory address, thus the size of a pointer is machine dependent
- With most data models it is the largest integer on the machine, size of **`unsigned long`**
- Defined in **`inttypes.h`**
  - **`uintptr_t`** and **`uintmax_t`**

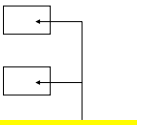
Section 5

Address Operations in C

- Declaration of pointer variables
  - The *pointer declarator* '**`*`**'
- Use of pointers
  - The *address of* operator '**`&`**'
  - The *indirection* operator '**`*`**' – also known as de-referencing a pointer

Pointer Declaration

- Syntax
  - *destinationType* **`*`** *varName*`;`
- Must be declared with its associated type.
- Examples
  - **`int *ptr1;`**
    A pointer to an **`int`** variable
  - **`char *ptr2;`**
    A pointer to a **`char`** variable



Pointers are NOT integers

- Although memory addresses are essentially very large integers, pointers and integers are not interchangeable.
- Pointers are not of the same type
- A pointer's type depends on what it points to
  - **`int *p1; // sizeof(int)`**
  - **`char *p2; // sizeof(char)`**
- C allows free conversion btw different pointer types via casting (dangerous)

Address of Operator

- Syntax
  - `&` *expression*

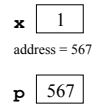
    The expression must have an address. E.g., a constant such as "`1`" does not have an address.
- Example
  - `int x = 1;`
    `f(&x);`

    The address of `x` (i.e. where `x` is stored in memory), say, the memory location 567, (not 1) is passed to `f`.

Pointer Assignment

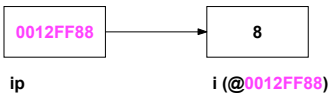
- A pointer `p` *points* to `x` if `x`'s address is stored in `p`
- Example
  - `int x = 1;`
    `int *p;`
    `p = &x;`

    Interpreted as:

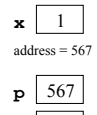
Pointer Diagram



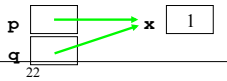

```
int i = 8;
int *ip;

ip = &i;
```


Pointer Assignment

- A pointer `p` *points* to `x` if `x`'s address is stored in `p`
- Example
  - `int x = 1;`
    `int *p, *q;`
    `p = &x;`
    `q = p;`

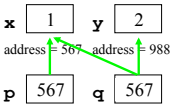
    

    Interpreted as:

Pointer Assignment

- Example
  - `int x=1, y=2, *p, *q;`
    `p = &x; q = &y;`
    `q = p;`

Indirection Operator

- Syntax
  - `*` *pointerVar*
  - Allows access to value of memory being pointed to
  - Also called *dereferencing*

Note: '`*`' in a declaration and '`*`' in an expression are different.
`int *p; int * p; int* p;`

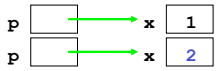
- Example
  - `int x = 1, *p;`
    `p = &x;`
    `printf("%d\n", *p);`

    `*p` refers to `x`; thus prints 1

Assignment Using Indirection Operator

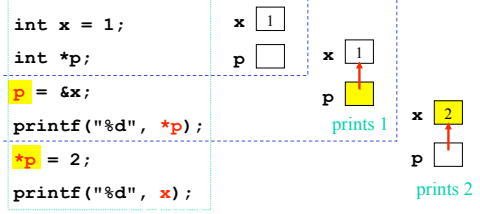
- Allows access to a variable indirectly through a pointer pointed to it.
- Pointers and integers are not interchangeable
- Example
  - `int x = 1, *p;`
    `p = &x;`
    `*p = 2;`
    `printf("%d\n", x);`



  - `*p` is equivalent to `x`

Schematically

```
int x = 1;
int *p;
p = &x;
printf("%d", *p);
*p = 2;
printf("%d", x);
```



Summary

- Pointer and integers are not exchangeable
- Levels of addressing (i.e. layers of pointers) can be arbitrarily deep
- Remember the `&` that you MUST put in front of `scanf` variables?
- Failing to pass a pointer where one is expected or vise versa always leads to segmentation faults.