And… what does it have to do with types and variables?

Look at it one way and it’s a wolf… Look at it another way and it’s just a bunch of characters.

It’s all in the way you look at it. …and think about it. …and change it. …and use it.

A bit is a binary digit: a 0 or a 1

- any data can be represented by enough bits
- bits are easy to represent in hardware
- bits are an incredible pain to deal with…

The information in the bits is all in how we (and the computer) look at them!

Today’s Outline

Memory structure of computers
Types
Variables and identifiers
Assignment statements
Tracing programs

Review:
Computer Organization
Memory

Memory is a collection of locations.
Each location is a group of bits.
To make use of these we need:

- a way of interpreting a location
  We use types to do this!
- a way to reference locations of interest
  We give the locations names (identifiers),
  and use these names to refer to them.

Tools: Types

A type is a way of interpreting a memory location
- describes the kind of information it can contain
- affects the way we can operate on it

Basic types include
- integers: whole numbers: 17, -42 “int” in C
- real numbers: 3.14159, 6.02e23 “double” in C
- character data: ’a’, ’?’, ’N’, ’’, ’9’ “char” in C

Type Example

```
<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>01000011</td>
<td>01010011</td>
<td>00110001</td>
<td>00110100</td>
<td>00110010</td>
</tr>
<tr>
<td>01000110</td>
<td>01010011</td>
<td>00110001</td>
<td>00110010</td>
<td>00110010</td>
</tr>
</tbody>
</table>
```

```
Memory

as a char
01000011
C

as an int
128’s
8’s
4’s
2’s
10001110
142
```

ASCII Table

ASCII (American Standard Code for Information Exchange)
defines the most common char interpretation for bits.

```
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>00111111</td>
</tr>
<tr>
<td>64</td>
<td>01000000</td>
</tr>
<tr>
<td>65</td>
<td>01000001</td>
</tr>
<tr>
<td>66</td>
<td>01000101</td>
</tr>
<tr>
<td>67</td>
<td>01000111</td>
</tr>
<tr>
<td>68</td>
<td>01001001</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Identifiers

(a fancy word for “names”)

"Identifiers” let us name memory locations
(and lots of other things! more later…)
Using these names we can refer to the contents of memory locations.

```
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000</td>
<td>01010011</td>
<td>00110001</td>
<td>00110010</td>
</tr>
<tr>
<td>00000001</td>
<td>00110100</td>
<td>10001110</td>
<td></td>
</tr>
</tbody>
</table>
```

Rules: Variables and Types in C

Variable declarations in C (set aside location)

```
<type> <name>;
```

int count;
double gasPrice;
char bang;

```
Picture:
```

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>count</td>
<td>gasPrice</td>
<td>bang</td>
</tr>
<tr>
<td>(int)</td>
<td>(double)</td>
<td>(char)</td>
</tr>
</tbody>
</table>
```

What’s missing that would tell us the size of these locations?
Initialization Values

Variable declarations in C (set aside location)

\[
\begin{align*}
\text{<type> <name> = <initial_value>;}
\end{align*}
\]

- `int count = 12;`
- `double gasPrice = 1.799;`
- `char bang = '!';`

```
<table>
<thead>
<tr>
<th>count</th>
<th>gasPrice</th>
<th>bang</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>1.799</td>
<td>'/'</td>
</tr>
</tbody>
</table>
```

Types in C

- `int months;`
  - “Integer” variables represent whole numbers:
    - 1, 17, -32, 0
  - Not 1.5, 2.6, ‘A’

- `double student_gpa;`
  - Floating point variables represent real numbers:
    - 3.14, -27.5, 6.022e23, 5.0
  - Not 5, ‘A’

- `char middle_initial, y_or_n_answer;`
  - Represent individual keyboard characters:
  - Not “Bill”, 0

Identifier Rules

Your book covers this beautifully! Read it.

Briefly, identifiers:
- contain only letters, digits, and underscore (‘_’)
- do not start with digits
- cannot be “reserved words” (like int)
- are “cAsE-SEnSitIVe”

One way to describe C identifiers is by “grammar rules”:

- letter \( \Rightarrow \) a or b or … or z or A or … or Z or _
- digit \( \Rightarrow \) 0 or 1 or … or 9
- identifier \( \Rightarrow \) letter (letter or digit) (letter or digit) …

The Way: Identifiers (1 of 2)

Any sequence of letters and numbers starting with a number is a valid identifier:

- `q, thx1138, _a_Random_varaibel`

But, not every sequence of letters and numbers is an equally good identifier!

You need to understand, remember, and type them.

So do others reading your code!

The Way: Identifiers (2 of 2)

The Way of variables means:
- use meaningful names: c vs. count
- name with a descriptive noun
- don’t use similar variable names:
  - never num_parts and num_parks
- most important: be consistent!
  - never, ever numParts and num_parts

Are we exempt? No!
- if you find us straying from the way, say so

But what do variables DO?

We can now declare variables, but how do we use them?

There are two things we might want to do with a variable:
- “access” (find out) its value
  - int i = 12; int j = 22
- set its value
  - int i = 37;
Assignment Statements

\[ \text{total} = \text{first\_part} + \text{second\_part}; \]

An assignment statement does both of these:
- the expression on the right is evaluated (formula of #s and vars.) (calculated out)
- evaluating a variable on the right accesses its value
- the variable on the left is set: its value becomes the value from the right

Total Example

\[
\begin{align*}
\text{double total} &= 1.0; \\
\text{double first\_part} &= 2.5; \\
\text{double second\_part} &= 2.0; \\
\text{total} &= \text{first\_part} + \text{second\_part};
\end{align*}
\]

Variables Everywhere:

\[
\text{my\_age} = \text{my\_age} + 1;
\]

How can that be read?
- “My age is equal to my age plus one.”
  \[\text{That’s impossible!}\]
  Fortunately, it’s also not what this really says.
- “Set my age to its current value plus 1.”
  \[\text{Ah… that’s much better, I believe in life again!}\]
  Assignments calculate the right side and store the result on the left. It’s not like algebra!
  So, the same variable can appear on both sides³

Some Examples in MSVC

This space accidentally left blank.

Putting It All Together:
Sequential Execution

First, all variables are given memory locations
- each variable declaration reserves (sets aside) a location
- adherents of The Way use names that Make Sense
Next, program execution begins.
“Control” of the CPU flows from one statement to the next.
Each statement is executed in sequence, one at a time.

An Example

\[
\begin{verbatim}
/* calculate and print area of 10x3 rectangle */
#include <stdio.h>
int main(void) {
  int rectangle_length; /* stores length */
  int rectangle_width; /* stores width */
  int rectangle_area; /* stores result (area) */
  rectangle_length = 10;
  rectangle_width = 3;
  rectangle_area = rectangle_length * rectangle_width ;
  printf("%d", rectangle_area);
  return 0;
}
\end{verbatim}
\]
Hand Simulation (Trace)

A useful practice is to simulate by hand the operation of the program, step by step.

This program has three variables, which we can depict by drawing boxes or making a table.

We mentally execute each of the instructions, in sequence, and refer to the variables to determine the effect of the instruction.

---

Tracing the Program

<table>
<thead>
<tr>
<th>rectangle_length</th>
<th>rectangle_width</th>
<th>rectangle_area</th>
</tr>
</thead>
<tbody>
<tr>
<td>after declaration</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>after statement 1</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>after statement 2</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>after statement 3</td>
<td>F</td>
<td>T</td>
</tr>
</tbody>
</table>

---

Initializing Variables

Initialization means giving something a value for the first time.

Anything which changes the value of a variable is a potential way of initializing it.

| initial value in a declaration: int i = 7; |
| assignment statement: count = 0; |

---

Initialization Rule

General rule: variables have to be initialized before their value is used.

Failure to initialize...

- is a common source of bugs
- is a semantic error, not a syntax error

*Why is this? What’s the problem? What might variables “start” as?*

---

Declaring vs Initializing

```c
int main (void) {
    double income; /* declaration of income, not an assignment or initialization */
    income = 35500.00; /* assignment to income, initialization of income, not a declaration */
    printf ("Old income is \$", income);
    printf ("Before raise: \$", income);
    income = 39000.00; /* assignment to income, not a declaration or initialization */
    printf ("After raise: \$", income);
    return 0;
}
```
Algorithm (result of analysis):
Celsius = 5/9 (Fahrenheit - 32)

What kind of data (result of analysis):
double fahrenheit, celsius;

Example Problem:
Fahrenheit to Celsius

Problem (specified):
Convert Fahrenheit temperature to Celsius

Fahrenheit to Celsius (I)
An actual C program

#include <stdio.h>
int main(void)
{
double fahrenheit, celsius;

celsius = (fahrenheit - 32.0) * 5.0 / 9.0;

return 0;
}

Fahrenheit to Celsius (II)

#include <stdio.h>
int main(void)
{
double fahrenheit, celsius;
printf("Enter a Fahrenheit temperature: ");
scanf("%lf", &fahrenheit);
celsius = (fahrenheit - 32.0) * 5.0 / 9.0;
printf("That equals %f degrees Celsius.", celsius);
return 0;
}

Running the Program

Enter a Fahrenheit temperature: 45.5
That equals 7.500000 degrees Celsius

Program trace

<table>
<thead>
<tr>
<th>Program trace</th>
<th>fahrenheit</th>
<th>celsius</th>
</tr>
</thead>
<tbody>
<tr>
<td>after declaration</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>after first printf</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>after scanf</td>
<td>45.5</td>
<td>?</td>
</tr>
<tr>
<td>after assignment</td>
<td>45.5</td>
<td>7.5</td>
</tr>
<tr>
<td>after second printf</td>
<td>45.5</td>
<td>7.5</td>
</tr>
</tbody>
</table>

Assignment step-by-step

celsius = (fahrenheit-32.0) * 5.0 / 9.0

1. Evaluate right-hand side
   a. Find current value of fahrenheit
      72.0
   b. Subtract 32.0
      40.0
   c. Multiply by 5.0
      200.0
   d. Divide by 9.0
      22.2

2. Assign 22.2 to be the new value of celsius
   (the old value of celsius is lost.)

Fahrenheit to Celsius (III)

#include <stdio.h>
int main(void)
{
double fahrenheit, celsius;
printf("Enter a Fahrenheit temperature: ");
scanf("%lf", &fahrenheit);
celsius = fahrenheit - 32.0;
celsius = celsius * 5.0 / 9.0;
printf("That equals %f degrees Celsius.", celsius);
return 0;
}
Does Terminology Matter?

Lots of new terminology today!
  - Variable, type, reserved word, initialization, declaration,
    statement, assignment, etc., etc.
You can write a complicated program without using
these words…
But you can’t talk about your programs without
them!
Learn the exact terminology as you go, and get in the
habit of using it.

Next Lecture: Expressions

Each lecture builds on the previous ones, so…
be sure you’re solid with this material
before going on!

QOTD: the Good, the Bad, and
the Ugly

For each of the situations
on the right, give a
variable name that is…
Good: legal, follows the Way
Bad: illegal
Ugly: legal, strays from the
Way
A variable to store the
user’s middle initial.
A variable that stores the
number of times the
user hits the ‘*’ key.
A variable that stores the
URL of your section’s
home page.

Can you make these creative, funny, or subtle?