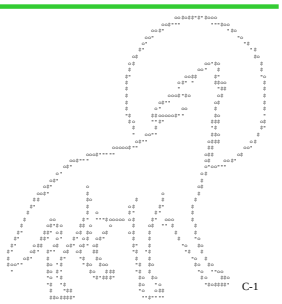
What is this?



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

C-1

CSE142
Computer Programming I

Variables

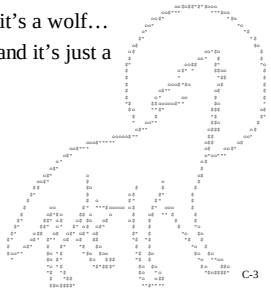
Or… making the best of a bunch of bits.

C-2

What is this?

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.

C-3

Computers Store Bits (and that's 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…

0 1 0 0 0 0 1 1	C
0 1 0 1 0 0 1 1	S
0 1 0 0 0 1 0 1	E
0 0 1 1 0 0 0 1	1
0 0 1 1 0 1 0 0	4
0 0 1 1 0 0 1 0	2

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

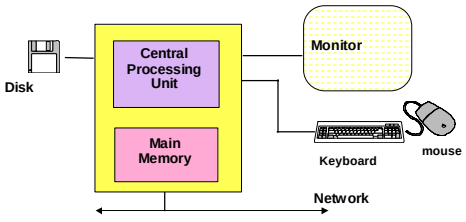
C-4

Today's Outline

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

C-5

Review: Computer Organization



C-6

Memory

Memory is a collection of locations

Each location is a group of bits

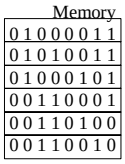
To make use of these we need:

– 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

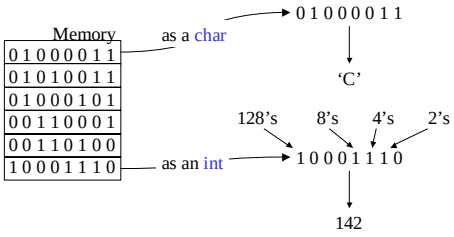
Basic types include

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



ASCII Table

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

A snippet from the ASCII table.

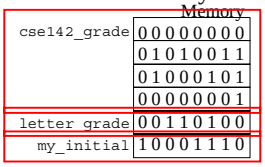
63	00111111	?
64	01000000	@
65	01000001	A
66	01000010	B
67	01000011	C
68	01000100	D

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.



What's missing that would tell us the ***size*** *of these locations?*

Rules: Variables and Types in C

Variable *declarations* in C (set aside location)

<type> <name>;

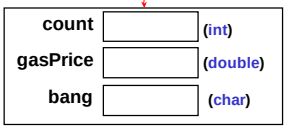
int count;

double gasPrice;

char bang;

What's missing from this picture?

Picture:



Initialization Values

Variable *declarations* in C (set aside location)

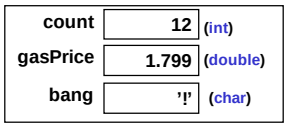
<*type*> <*name*> = <*initial_value*>;

int count = 12;

double gasPrice = 1.799;

char bang = '!';

Picture:



C-13

Types in C

int months;

"Integer" variables represent whole numbers:

1, 17, -32, 0 *Not 1.5, 2.0, '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:

'a', 'b', 'M', '0' , '9' , '#' , ' ' *Not "Bill", 0*

C-14

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 ⇒ **a** or **b** or … or **z** or **A** or … or **Z** or _

digit ⇒ **0** or **1** or … or **9**

identifier ⇒ **letter** (**letter** or **digit**) (**letter** or **digit**) …

C-15

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!

C-16

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, ever numParts and num_parts

Are we exempt? No!

- if you find us straying from the way, say so

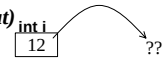
C-17

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



set its value

C-18

Assignment Statements

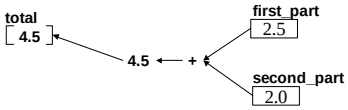
total = first_part + second_part;

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

```
double total = 1.0;
double first_part = 2.5;
double second_part = 2.0;
total = first_part + second_part;
```



Variables Everywhere: my_age = my_age + 1;

How can that be read?

"My age is equal to my age plus one."

That's impossible!
Fortunately, it's also not what this really says.

"Set my age to its current value plus 1."

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 sides!

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. *…for now.*

An Example

```
/* 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;
}
```

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

	rectangle_length	rectangle_width	rectangle_area
after declaration	?	?	?
after statement 1	10	?	?

Tracing the Program

	rectangle_length	rectangle_width	rectangle_area
after declaration	?	?	?
after statement 1	10	?	?
after statement 2	10	3	?
after statement 3	10	3	30

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

```
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 %f", income);
    income = 39000.00;       /* assignment to income  not a
                               declaration,or initialization */
    printf ("After raise: %f", income);

    return 0;
}
```

Example Problem: Fahrenheit to Celsius

Problem (specified):
Convert Fahrenheit temperature to Celsius

Algorithm (result of analysis):

Celsius = 5/9 (Fahrenheit - 32)

What kind of data (result of analysis):

double fahrenheit, 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;
}
```

Any problems?

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

	fahrenheit	*celsius*
after declaration	?	?
after first *printf*	?	?
after *scanf*	45.5	?
after assignment	45.5	7.5
after second *printf*	45.5	7.5

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
   b. Multiply by 5.0 — 200.0
   c. 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?