University of Washington Computer Programming I

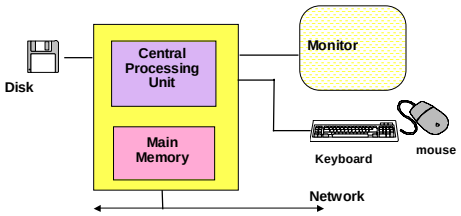
Lecture 3:
Variables, Values, and Types

Overview

Concepts this lecture:

- Variables
- Declarations
- Identifiers and Reserved Words
- Types
- Expressions
- Assignment statement
- Variable initialization

Review: Computer Organization



Review: Memory

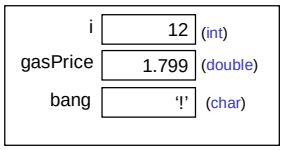
- Memory is a collection of locations
- Within a program, the locations are called **variables**
- Each variable has
  - A **name** (an **identifier**)
  - A **type** (the kind of information it can contain)
- Basic types include
  - **int** (integers – whole numbers: 17, -42)
  - **double** (floating-point numbers with optional fraction and/or exponent: 3.14159, 6.02e23)
  - **char** (character data: 'a', '?', 'N', ' ', '9')

Memory example

Variable declarations in C

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

Picture:



Declaring Variables

```
int months;
```

Integer variables represent whole numbers:
1, 17, -32, 0 Not 1.5, 2.0, 'A'

```
double pi;
```

Floating point variables represent real numbers:
3.14, -27.5, 6.02e23, 5.0 Not 3

```
char first_initial, middle_initial, marital_status;
```

Character variables represent individual keyboard characters:
'a', 'b', 'M', '0' , '9' , '#' , ' ' Not "Bill"

Variable Names

"Identifiers" are names for things in a program
- for examples, names of variables

In C, identifiers follow certain rules:
- use letters, numerals, and underscore (_)
- do not begin with a numeral
- cannot be "reserved words"
- are "case-sensitive"
- can be arbitrarily long but...

Style point: Good choices for identifiers can be extremely helpful in understanding programs
- Often useful: noun or noun phrase describing variable contents

Reserved words

Certain identifiers have a "reserved" (permanent, special) meaning in C
- We've seen **int** already
- Will see a couple of dozen more eventually

These words always have that special meaning, and cannot be used for other purposes.
- Cannot be used names of variables
- Must be spelled exactly right
- Sometimes also called "keywords"

Under the Hood

All information in the CPU or memory is actually a series of 'bits': 1's and 0's
- Known as 'binary' data
- Amazingly, all kinds of data can be represented in binary: numbers, letters, sounds, pictures, etc.

The type of a variable specifies how the bits are interpreted

Binary	C type	(sample) value
01010001	int	161
	char	'A'
	double	10.73

Normally we ignore the underlying bits and work with C types

Assignment Statements

An **assignment statement** stores a value into a variable.
The assignment may specify a simple value to be stored, or an **expression**

```
int area, length, width;      declaration of 3 variables */
length  = 16;                 /* "length gets 16"        */
width   = 32;                 /* "width gets 32"         */
area    = length * width;     /* "area gets length times width" */
```

Execution of an assignment statement is done in two distinct steps:
- Evaluate the expression on the right hand side
- Store the value of the expression into the variable named on the left hand side

my_age = my_age+1

This is a "statement", not an equation. Is there a difference?
The same variable may appear on <u>both</u> sides of an assignment statement

```
my_age = my_age + 1 ;
balance = balance + deposit ;
```

The **old** value of the variable is used to compute the value of the expression, <u>before</u> the variable is changed.
You wouldn't do this in algebra!

Program Execution

A memory location is reserved by declaring a C variable
- You should give the variable a name that helps someone else reading the program understand what it is used for in that program

Once all variables have been assigned memory locations, program execution begins

The CPU executes instructions one at a time, in order of their appearance in the program (we will introduce more options later)

An Example

```
/* calculate and print area of 10x3 rectangle */
#include <stdio.h>
int main(void  {
    int    rectangleLength;
    int    rectangleWidth;
    int    rectangleArea;
    rectangleLength = 10;
    rectangleWidth = 3;
    rectangleArea = rectangleLength * rectangleWidth ;
    printf("%d", rectangleArea);
    return 0;
}
```

C-13

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

C-14

Tracing the Program

	rectangleLength	rectangleWidth	rectangleArea
after declaration	?	?	?
after statement 1	10	?	?

C-15

Tracing the Program

	rectangleLength	rectangleWidth	rectangleArea
after declaration	?	?	?
after statement 1	10	?	?
after statement 2	10	3	?
after statement 3	10	3	30

C-16

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.

- For now, that means assignment statement

C-17

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

Variables in a C program are not automatically initialized to 0!

C-18

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

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;
}
```


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", "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!