CSE / ENGR 142 Programming I

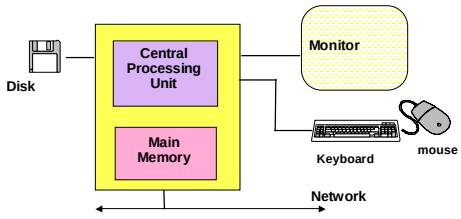
Variables, Values, and Types

© 1998 UW CSE

Chapter 2 Overview

- Chapter 2: Read Sections 2.1-2.6, 2.8.
  - Long chapter, short snippets on many topics
  - Later chapters fill in detail
- Specifically:
  - Types, variables, values
  - Expressions, assignment
  - Input / Output (*scanf*, *printf*)
  - Programming style
- ***You'll learn enough to write a simple but useful C program!***

Review: What's a Computer?



Memory and Data

- All data created and manipulated by a program is stored in the computer's memory
- We think of memory as being organized as a sequence of words.
- Each word has three attributes:
  - its location or address (its "name")
  - its value (the data stored in that location)
  - its type (the kind of value stored there)

Memory and Data

Location	Value
6:	'm'
5:	3.1416
4:	-112
3:	214
2:	'y'
1:	3

- location 5 holds the value 3.1416
- location 3 holds the value 214

Variables

- If we had to name memory locations explicitly by number we'd go crazy, so...
- In our programs we name memory symbolically, by defining variables.
- Each variable must be declared.
- The declaration indicates the name and the type of the variable.
  - The type indicates how to interpret the value stored there.

Declaring Variables

```
int months;
```

Integer variables represent whole numbers:

1, 17, -32, 0 Not 1.5, 2.0

```
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"

Memory, Variables, Values, and Types

Location	Value	Type	Possible variable name
6:	'm'	*char*	*first_initial*
5:	3.1416	*double*	*almost_pi*
4:	-112	*int*	*my_balance*
3:	214	*int*	*your_balance*
2:	'y'	*char*	*answer*
1:	3	*int*	*Image3D*

Identifiers

- "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 words have a "reserved" (permanent, special) meaning in C
  - We've seen *int*, *double*, *char* 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"

Declarations vs Statements

- Programs are made up of "declarations" and "statements"
- Declarations declare or define something
  - We've already seen variable declarations
  - Later we'll see constant declarations and function declarations
- Statements tell the CPU to do something
  - We're about to see our first kind of statement
  - Eventually we'll see about a dozen other kinds of statements

Storing values in variables

- A variable declaration gives a name to a memory location, but... *how do we place a value in that memory location?*
- Placing a value in a location is called "storing"
- One way to store a value is with the assignment statement.
  - Later we'll see that *scanf* can also store values into a variable.

Assigning Values

- An assignment statement places 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" */
```

- *The result: store the value of the expression on the right into the variable on the left.*

C Program Structure

```
#include <stdio.h>
int main(void)
{

    variable declarations

    program statements


    return(0);
}
```

The shaded parts will be in *every* program (we'll explain later)

Problem Solving and Program Design (Review)

- Clearly specify the problem
- Analyze the problem
- Design an algorithm to solve the problem
- Implement the algorithm (write the program)
- Test and verify the completed program
  - The test-debug cycle
- Maintain and update the program

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

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);
}
```


Assignment step-by-step

*celsius = celsius * 5.0 / 9.0 ;*

1. Evaluate right-hand side
   a. Find current value of celsius 13.5
   b. Multiply by 5.0 67.5
   c. Divide by 9.0 7.5
2. Assign 7.5 to be the new value of celsius
   (old value 13.5 of celsius is lost.)

my_age = my_age+1

- The same variable may appear on both 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, before the variable is changed.
- *You wouldn't do this in math!*

Note on lecture examples

- Slides often leave out important details
  my_age = my_age + 1;
- This is a legal C statement only if:
  - my_age has previously been declared in the program
  - my_age has a proper type (e.g. *int*)
  - the statement occurs in a legal position;
  - the full program has "*int main (void)*", etc., etc.
- Use your creative powers and common sense to deduce what's missing in the examples!

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
- *General rule: variables have to be initialized before their value is used.*
  - Failure to initialize is a common source of bugs.
- Variables in a C program are not automatically initialized to 0!

Does Terminology Matter?

- "variable", "reserved word", "initialization", "declaration", "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.
  - Your TAs, consultants, and tutors will bless you…
  - … and will be able to better help you

Declaring vs Initializing

```
int main (void) {
        double income;          /*declaration of income,
                                  not an assignment,
                                  not an 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,
                                  not an initialization */
        printf ("After raise: %f", income);
}
```


Compilers, Linkers, etc.