

CSE 142 Programming I

Variables, Values, and Types

© 2000 UW CSE

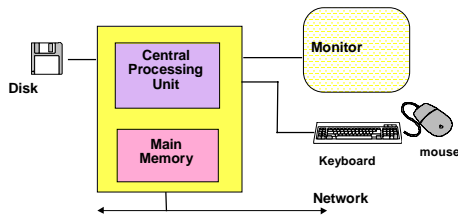
1/5/00 B-1

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!*

1/5/00 B-2

Review: What's a Computer?



1/5/00 B-3

Inside the CPU and Memory

- We've talked about what the CPU does
 - Executes instructions one at a time
 - Series of instructions constitute "programs"
- The memory holds information for use by the CPU
 - Organized as a numbered series of "locations"
 - Each location holds one unit of information
- All information in the CPU or memory is 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.

1/5/00 B-4

Memory

Address	Contents
0:	01101110
1:	00000000
2:	00000001
3:	10001000
4:	11111111
5:	01110111
6:	00010110

A Program (CPU Instructions)

1. Set location 4 to 00000001
2. Set location 5 to 00000010
3. Add the contents of locations 4 and 5 and put the result in location 2
4. Print the contents of location 2 as an integer

1/5/00 B-5

Variables

- If programmers had to do everything in binary... *they would go crazy!*
- If programmers had to remember the memory locations of the data... *they would go crazy!*
- Fortunately, programming languages give you a way around these details:
 - a "variable" is a name for a location in memory.
 - variables have "types," which lets us think about the values in human rather than binary terms
- Puzzle: why do programmers still go crazy?

1/5/00 B-6

How to Say It in C

```
#include <stdio.h>
```

```
int main(void) {
```

```
    int    firstOperand;  
    int    secondOperand;  
    int    thirdOperand;
```

```
    firstOperand = 1;  
    secondOperand = 2;  
    thirdOperand = firstOperand + secondOperand;  
    printf("%d", thirdOperand);
```

```
    return 0;  
}
```

Key

- Stuff you need in any C program
- Memory allocation ("Declarations of variables")
- Directions for CPU ("Executable instructions" or "C statements")

1/5/00 B-7

Memory

Address	Contents
0:	01101110
1:	00000000
(thirdOperand) 2:	00000001
3:	10001000
(firstOperand) 4:	11111111
(secondOperand) 5:	01110111
6:	00010110

A Program (CPU Instructions)

- Set location 4 (firstOperand) to 00000001 (decimal 1)
- Set location 5 (secondOperand) to 00000010 (decimal 2)
- Add the contents of locations 4 and 5 and put the result in location 2 (thirdOperand)
- Print the contents of location 2 (thirdOperand) as an integer

1/5/00 B-8

Important Points

- A memory location is reserved by declaring a C variable
- You can 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
- Instructions are executed one at a time, in order of their appearance in the program
- You should *initialize* variables before trying to use their values

1/5/00 B-9

Another Example

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

```
    int    rectangleLength;  
    int    rectangleWidth;  
    int    rectangleArea;
```

```
    rectangleLength = 10;  
    rectangleWidth = 3;  
    rectangleArea = rectangleLength * rectangleWidth ;  
    printf("%d", rectangleArea);
```

```
    return 0;  
}
```

1/5/00 B-10

"Hand Simulation"

1/5/00 B-11

In a Little More Depth

- Declarations:
 - Choosing variable names
 - Reserved words
 - Variable *types*
- The *assignment statement*

1/5/00 B-12

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

1/5/00 B-13

Examples

OK	Illegal	Legal, But Not OK
rectangleWidth	10TimesLength	a1
rectangle_Width	My Variable	1
rectangle_width	int	O
length_10_Rectangle		rectangleWidth and rectanglewidth or rectangle_Width

1/5/00 B-14

Reserved words

- Certain words 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"

1/5/00 B-15

"Types"

- Each C variable names a memory location in the computer
- Each memory location contains a set of bits (0's and 1's)
- The value the 0's and 1's represent in the C program depend on the *type* of the variable
- Examples of three C types (all we'll see for quite a while)

Binary	C Variable Type	(Example)Value
01010001	int	161
	char	'A'
	double	10.73

1/5/00 B-16

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"
```

1/5/00 B-17

Assignment Statements

- 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" */
```

- **Operation: CPU will store the value of the expression on the right into the variable on the left.**

1/5/00 B-18

my_age = my_age+1

- This is a "statement", not an equation. Is there a difference?

- 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!*

1/5/00 B-19

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!

1/5/00 B-20

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);
}
```

1/5/00 B-21

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

1/5/00 B-22

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;
```

1/5/00 B-23

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);
}
```

1/5/00 B-24

Fahrenheit to Celsius (II)

```
#include <stdio.h>

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

1/5/00 B-25

Running the Program

Enter a Fahrenheit temperature: **45.5**
That equals 7.500000 degrees Celsius

Program "trace:"

	<u>fahrenheit</u>	<u>celsius</u>
after declaration	?	?
after first <i>printf</i>	?	?
after <i>scanf</i>	45.5	?
after assignment	45.5	7.5
after second <i>printf</i>	45.5	7.5

1/5/00 B-26

Fahrenheit to Celsius (III)

```
#include <stdio.h>

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

1/5/00 B-27

Assignment step-by-step

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

- Evaluate right-hand side
 - Find current value of **fahrenheit** **72.0**
 - Subtract **32.0** **40.0**
 - Multiply by **5.0** **200.0**
 - Divide by **9.0** **22.2**
- Assign **22.2** to be the new value of **celsius**
(any old value of **celsius** is lost.)

1/5/00 B-28

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!

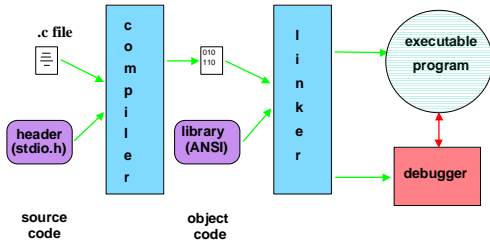
1/5/00 B-29

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.
 - Your TAs, consultants, and tutors will bless you...
 - ... and will be able to better help you

1/5/00 B-30

Compilers, Linkers, etc.



1/5/00 B-31