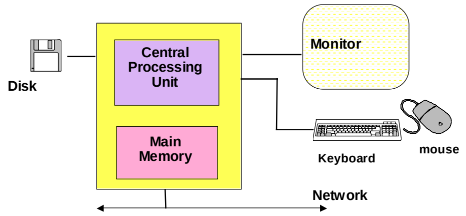
CSE 142
Programming I

Variables, Values, and Types

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?



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.

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*

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?

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”)

Memory

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

A Program
(CPU Instructions)

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

Important Points

1. A memory location is reserved by declaring a C variable
2. You can give the variable a name that helps someone else reading the program understand what it is used for in that program
3. Once all variables have been assigned memory locations, program execution begins
4. Instructions are executed one at a time, in order of their appearance in the program
5. You should *initialize* variables before trying to use their values

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;
}
```

“Hand Simulation”

In a Little More Depth

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

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 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”

“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

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"

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.*

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!*

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!

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);
}
```

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 = (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

(any old value of celsius is lost.)

Note on lecture examples

- Slides often leave out important details

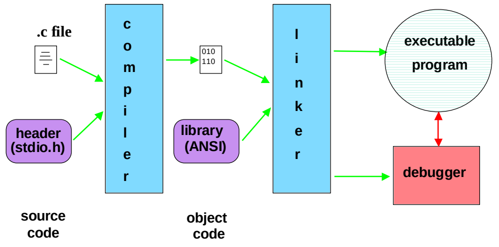
  my_age = my_age + 1;

- 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!

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

Compilers, Linkers, etc.


.c file
header (stdio.h)
source code
compiler
010
110
library (ANSI)
object code
linker
executable program
debugger