Lecture 2:

First Program, Data Types, Arithmetic Operators, and Basic I/O

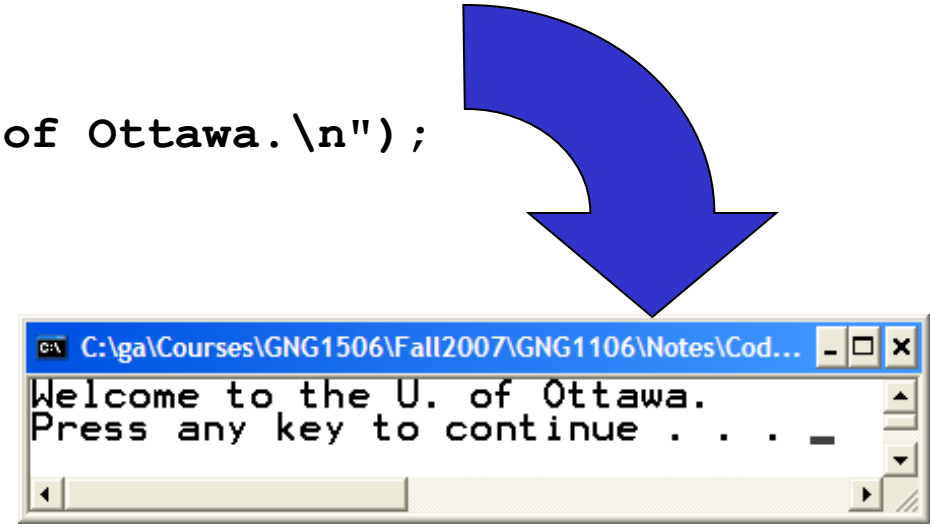
Prof. Shervin Shirmohammadi
University of Ottawa

Our First C Program

- The following simple program prints out a message to the screen:

```
/*****************************************
Our first program in C.
Print out a message to the screen.
*****************************************/
#include <stdio.h>

void main()
{
   printf("Welcome to the U. of Ottawa.\n");
}
```

- The programming environment on my PC running Windows XP, creates a window containing the output on the right:



- **Comment lines.**
  - All lines that begin with `/*` and end with `*/` are comments.
  - Anything written between this pair is ignored by the compiler and does not generate any executable code.

    ```
    /* The comment is placed here. */
    ```

    - This spans multiple lines, if needed.
  - A comment may also be written using `//` placed anywhere on a line: anything written to the right of the `//` will be ignored by the compiler; this is actually a C++ command.

    ```
    // The comment is placed here.
    ```

    - This covers only one line.
  - Comments are written for humans and are considered to be the internal documentation of the program; the external documentation is the software report associated with the program.
- **The line `#include<stdio.h>` is a preprocessor directive.**
  - All lines that begin with `#` are preprocessor directives which are processed before the compilation of the C code.
  - In this case, the directive `include` makes the compiler include a file named `stdio.h` into the compilation process. This file contains the standard C I/O (Input/Output) function headers and definitions.
  - Note that there is no `;` after a preprocessor directive.

- Function header `void main().`
  - The function name `main()` is required in all C programs.
  - All C programs begin executing after this line.
  - We shall study C function definitions in great detail; as we shall see, if `main()` is of type **`void`**, it returns nothing to the computer's operating system. If the parentheses `()` are empty, then the function does not receive arguments from the operating system.
- The left brace `{` indicates the beginning of `main()` and the right brace `}` indicates its end.
  - This defines the instruction bloc for the function
- The lines `printf("Welcome to the U. of Ottawa.\n");` calls the standard C I/O function `printf()` which prints out the desired message to the screen.
  - The message to be written is included in a pair of double quotes " and ".
  - `printf` is a function and the argument to the function is included in parentheses `()`.
  - `\n` is an escape sequence which positions the cursor at the beginning of the next line.
  - `printf` makes a call to the operating system
- The semi-colon `;` must end all C statements.
- C is case sensitive.
  - UPPERCASE letters are not the same as lowercase letters.

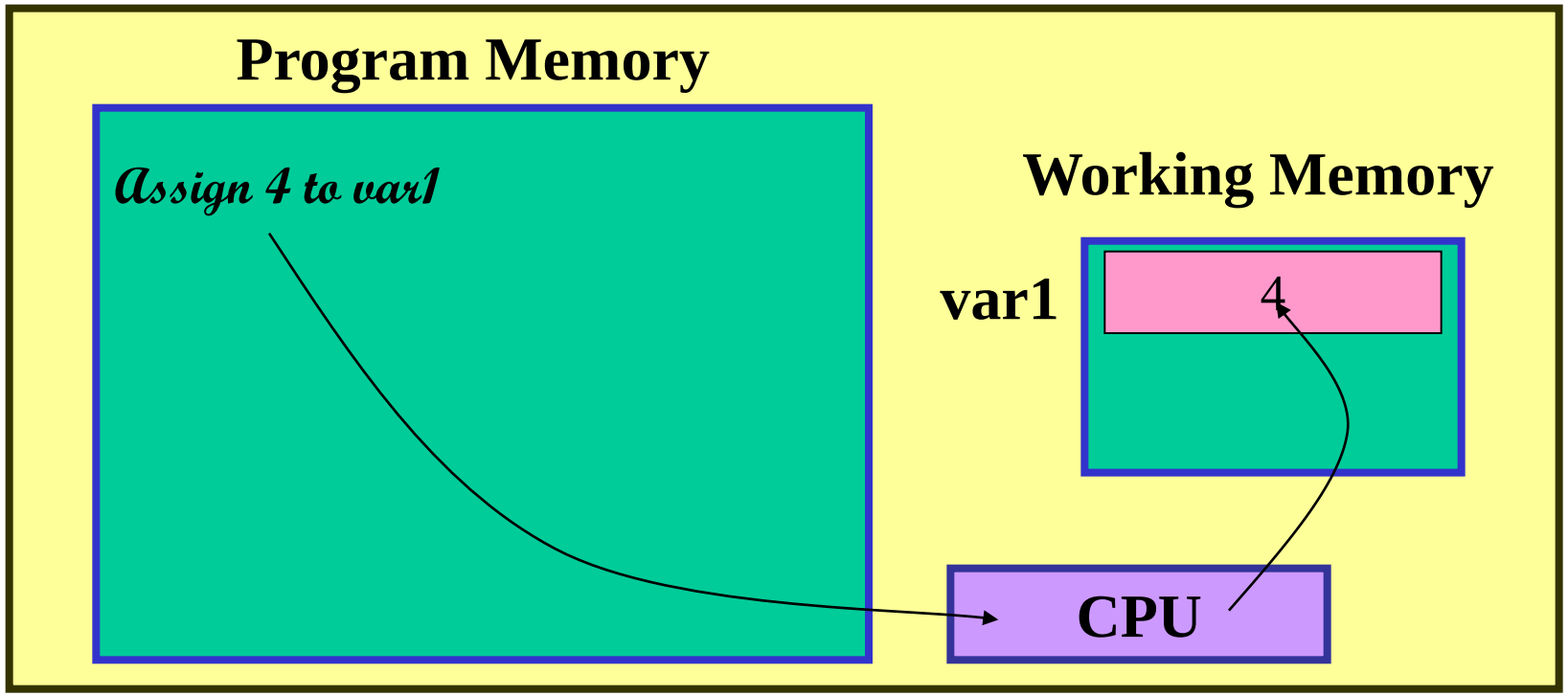
Variables

- The basic object in any computer language is the ***variable***
- A variable corresponds to **a location** in memory and has **an address**.
  - The type of variable defines how much memory it takes to store a value of its type, e.g. it takes less memory to store an integer value than a real value
- The variable name corresponds to the address of the variable (when a program is compiled, a name is translated to an address for use by the CPU to access the contents of the variable)
- E.g.

```
int x = 3;
int y;
float z = 5.6;
y = x * 5;
```

The Assignment Operation

- Values are assigned to variables, which means that a value is stored in the memory location reserved for the variable.
- The assignment operation is destructive, i.e. anything store in the variable is lost when a new value is assigned to the variable
- Consider the following pseudo-code: the CPU executes the instruction (in reality a number of CPU instructions) and stores the value 1 into the variable



Data Types in C

- The basic "objects" in C are variables and symbolic constants.
- The content of a variable can be changed by a program but a symbolic constant cannot be changed.
- All variables and symbolic constants **must be declared and defined before usage** in a program.

Names of Variables and Symbolic Constants

- All letters `a` to `z` and `A` to `Z`, all numbers `0` to `9` and the underscore `_` can be used in the name of a variable or symbolic constant.
- C is case sensitive so `a` is not recognized as being the same letter as `A`.

  E.g.: `STUDENT_1` is not the same name as `student_1`

- A variable name cannot begin with a number.
- C programmers usually follow the convention that variable names are written in lowercase letters while symbolic constants are written in UPPERCASE letters.
- Reserved words cannot be used as variable names.

  E.g.: `if`, `else`, `while`... are reserved words since they are used in C commands so these words cannot be used as variable names.
- **Use descriptive variable names!** It will make it much easier to remember what it means.

  E.g.: use `student_1` instead of `s1`

Declaring Variables

- Variables must be defined via declarations.
  - Declarations are usually placed after the first `{` of `main` (or of a function definition) and before the first command.
  - A declaration contains at least the variable name and its type.
- The type of a variable defines the nature or kind of data that can be stored in the variable.
- A variable consists of a space in memory where the program can store and retrieve data.

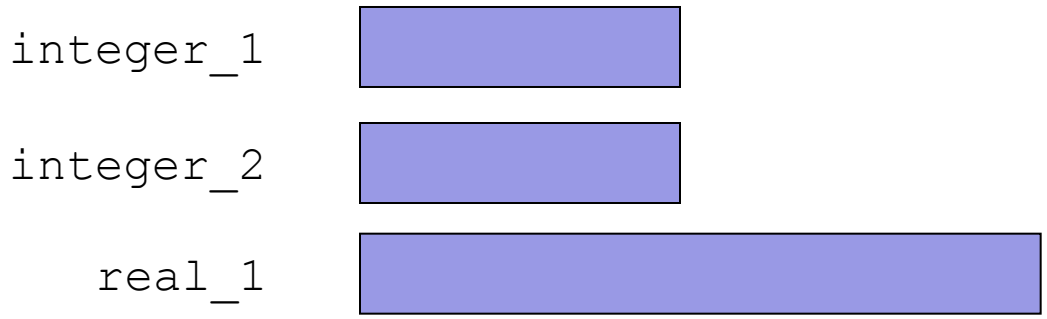
- There are 4 basic data types in C:

`char`	stores one character: `a` to `z`, `A` to `Z`, `!`, `$`,...
`int`	stores an integer: `1`, `101`, `-1462`,...
`float`	stores a real number: `0.5`, `-122.34`,...
`double`	stores a real number in double precision.

- Here are a few examples of variable declarations in C:

```
int integer_1, integer_2;
float real_1;
```

  - A declaration begins with the type followed by the variable names separated by commas `,` and ends with a semi-colon `;`
  - These declarations specify that the variables `integer_1` and `integer_2` are of type `int` and thus can store integer numbers, while `real_1` is a variable of type `float` that can store a real number. Space in memory will be allocated to these variables by the compiler:



Symbolic Constants

- "Magic numbers" sprinkled throughout a program is generally considered to be an example of bad programming style.

- If a "magic number" must be used in the program, then it is best to associate the number to a symbolic constant that has a descriptive name and then to use the constant in the program.

- Symbolic constants are defined using the pre-processor directive `define`, E.g.:

```
#define NBR_OF_STUDENTS 455
#define PI 3.141593
```

Define NBR_OF_STUDENTS as 455

Define PI as 3.141593

  - Pre-processor directives do not require a `;` at the end.

- The pre-processor performs a textual substitution of the symbolic constant name with the associated quantity (directly to the right) everywhere in the code before compilation.
  - ➔ No memory space is used for a symbolic constant. They are not variables.
  - ➔ A program cannot change the value associated with a symbolic constant.
  - ➔ Be careful with the `;`

    E.g.: `#define MAX 100;` *Define MAX as 100*

    will result in a substitution of `MAX` by `100;` everywhere in the program!

Arithmetic Operators and Expressions

- The = operator is the assignment operator: it assigns a quantity to a variable.
  - It is a binary operator; i.e., it requires 2 objects, one placed on either side.

    E.g.:
    ```
    integer_1 = 1;
    real_1 = 10.0;
    ```
  - The assignment operation is destructive. Any value previously stored in the variable will be erased and the new value will be assigned to the variable.
  - It is possible to make an assignment in a variable declaration.

    E.g.: `int integer_1 = 1;`

    defines `integer_1` as being a variable of type `int` and stores the value `1` in it.

- The following binary arithmetic operators exist in C:

Operation	C Binary Operator	Typical C Arithmetic Sub-expression
addition	+	a + b
subtraction	-	a - b
multiplication	*	a * b
division	/	a / b
modulus	%	a % b

Rules of Operator Precedence

- Brackets `()` are used to group sub-expressions in order to
  - make expressions more readable, and
  - change the order of precedence of the operators.

    E.g.: `e = (a + b) * (c + d);`

    which is different from : `e = a + b * c + d;`

- C evaluates arithmetic expressions according to the following rules of operator precedence.

1. Contents of parentheses are evaluated first.
   - If there are many "levels" of parentheses, then the innermost pair is evaluated first; the next innermost pair is evaluated second…
   - If there are many pairs of parentheses on the same level then they are evaluated left to right.
2. Negation (unary) is evaluated next.
3. Multiplications, divisions and moduli are evaluated next.
   - If there are many then they are evaluated from left to right.
4. Additions and subtractions are evaluated last.
   - If there are many, they are evaluated left to right.

Examples

```
e = a * (b * (c + d));
```

3 2 1

```
e = (a + b) * (c + d);
```

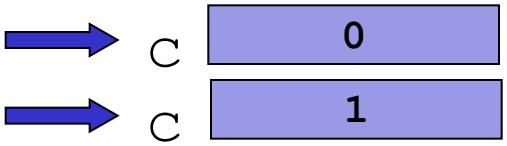
1 3 2

```
e = p * r % q + w / x - y;
```

1 2 4 3 5

Division of Integers

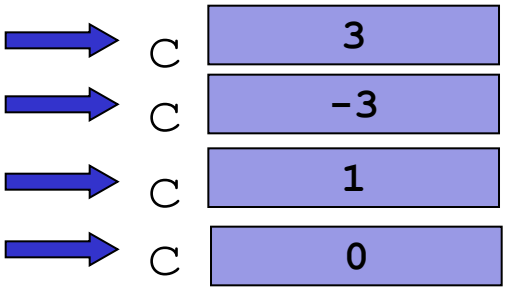
- Suppose that `a`, `b` and `c` are of type `int`.

E.g.:
```
a = 3;
b = 4;
c = a / b;
c = b / a;
```


  - The division of integers returns the quotient.

The Modulus

- Operation that computes the remainder of a division of integers. Suppose that `a`, `b` and `c` are of type `int`.

E.g.:
```
a = 3;
b = 4;
c = a % b;
c = -a % -b;
c = b % a;
c = b % b;
```


Sign of remainder is sign of numerator.

  - The modulus operates on integers only.

Mixed Type Arithmetic Expressions

- An expression that contains variables of many types will be converted in memory by the C compiler according to the implicit rule of promotion.

- The promotion hierarchy of our 4 basic data types is as follows:

  `double` ⟵ highest type

  `float`

  `int`

  `char` ⟵ lowest type

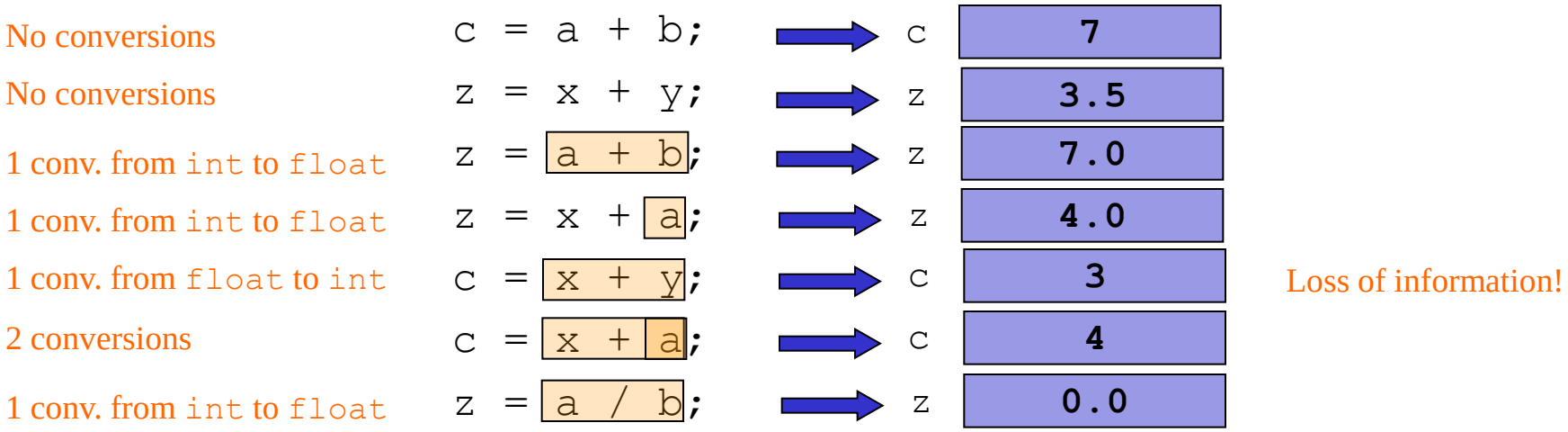
  – The hierarchy is organized according to the amount of information that can be stored in the data type.
    - A double variable can contain the same amount of information as all of the "lower" data types.

- The implicit rule of promotion followed by the C compiler when evaluating mixed type expressions is:
  – All variables in an expression are promoted to the highest type (in temporary memory) before evaluation of the expression.

- Consider the following examples of mixed type expressions.

E.g.:

```
int a, b, c;
float x, y, z;
a = 3;
b = 4;
x = 1.0;
y = 2.5;
```



- Care must be taken when storing a real number into an `int`. The real number will be **truncated** and only the integer portion will be stored.

- Careful with the division of integers; in the last example, `a / b` is evaluated as an integer division, the result `0` is promoted to a float `0.0` and stored in `z`.

The Cast Operator

- The unary `cast` operators can be used to force the type conversion of a variable.

  E.g.: (same variable definitions as in the previous example)

  `z = (float) a / b;` → `z` **0.75**

  - `(float) a` forces the conversion of `a` to type `float`,
  - according to the implicit rule of promotion, `b` is converted to type `float` too,
  - the division is then evaluated as a division of real numbers and the result is assigned to `z` which is of type `float`.

- A unary operator operates on one argument only: the argument directly to its right.

- A cast operator exists for each data type in C: `(double)`, `(float)`, `(int)`, `(char)`.

Summary of Hierarchy and Associativity of Arithmetic Operators

`()`	left to right
`* / %`	left to right
`+ -`	left to right
`=`	right to left

Basic Output

- Recall that a computer program interacts with the operating system
- To output a message to a screen, some call to the operating system is required.
- We shall represent such a call simply with the word ***Print*** as in

  Print "Welcome to the U. of Ottawa"

  – Note that the string to be printed is enclosed in quotes.
- To print the value of a variable use the following form of pseudo-code

  Print "The value of x is", x

  – If x contains the value 3, then the message "The value of x is 3" will be printed.

  – What message does the following print, given that x contains 4 and y contains 5?

  Print "The value of x is ", x, " and y is ", y

Basic Input

- To read a value from the keyboard, some other call to the operating system is required.
- Shall represent such a call simply with the word ***Read*** as in

  Read a value into var1

  – The value read from the keyboard is assigned to the variable *var1*.

Basic Output in C

- `printf` can be used to print out numbers and the content of variables to the screen:

```
E.g.:  printf("%d", 100);
       printf("%f", 101.3);
       printf("%d", integer_1);
       printf("%f", real_1);
```

- The `%d` is the conversion specifier used to print out an integer and the `%f` is the specifier used to print out a real number.
  - The conversion specifier informs `printf` of the type of the data to be printed.
  - A conversion specifier always begins with a `%`
  - The specifier must be enclosed in a pair of quotes " and "
  - As we shall see, there are many more conversion specifiers.
- More than one "object" can be written out using a single `printf`:

```
E.g.:  printf("The sum is %f", sum);
```

  - The arguments to printf are seperated by a comma `,`
- A `%` can be printed out as part of a message.
  - Use two percent symbols `%%` to print one out.

Escape Sequences

- The backslash `\` in this context is called the escape character and `\n` is called an escape sequence. The escape sequence is an instruction to the `printf` function. There are many escape sequences:

`\n`	positions the cursor at the beginning of the next line
`\t`	positions the cursor at the next tab stop
`\r`	positions the cursor at the beginning of the current line
`\a`	rings the bell (alarm)
`\\`	prints out the `\` symbol (backslash)
`\"`	prints out the " symbol (double-quote)

Basic Input in C

- Data can be entered from the keyboard using the standard C function `scanf`.

  E.g.:
  ```
  scanf("%d", &integer_1);
  scanf("%f", &real_1);
  scanf("%d%f", &integer_1, &real_1);
  ```

- `scanf` requires a conversion specifier corresponding to the type of data to be read.
- An ampersand `&` must be placed before the variable name into which the data will be read.

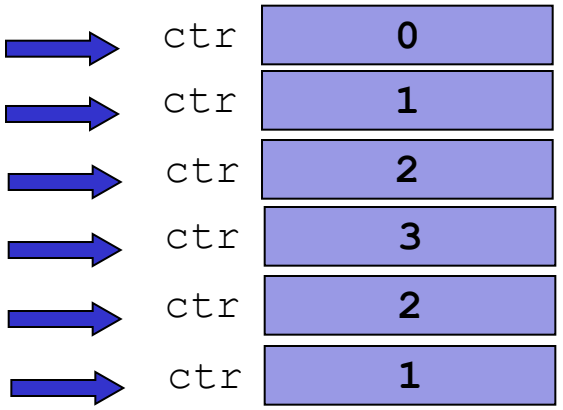
Standard Input and Output Header File

- **Information** related to the standard functions `printf` and `scanf` is contained in the standard input/output header file `stdio.h`.
- The pre-processor directive `#include <stdio.h>` placed before `main()` forces the compiler to include this information in the compilation process.

Counters

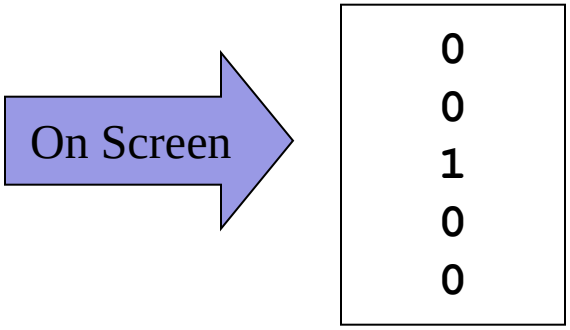
- Counters are often used in loops to keep track of the number of times that a certain task is done. A counter in C is usually a variable of type `int` that we increment or decrement:
  - E.g.: `int ctr = 0;`
    - `ctr = ctr + 1;` *Increment ctr*
    - `ctr = ctr - 1;` *Decrement ctr*

- There exists in C, special operators for adding or subtracting 1 from a variable; these operators are called the increment operator `++` and the decrement operator `--`.
  - If the increment or decrement operator is placed before the variable then it is called a pre-increment or pre-decrement operator.
  - If the increment or decrement operator is placed after the variable then it is called a post-increment or post-decrement operator.
    - E.g.:
      - `int ctr = 0;`
      - `ctr = ctr + 1;`
      - Post-increment `ctr++;`
      - Pre-increment `++ctr;`
      - Post-decrement `ctr--;`
      - Pre-decrement `--ctr;`



- Careful with the use of the ++ and -- operators.
  - The pre-increment operator adds 1 to a variable before using it in an expression.
  - The post-increment operator adds 1 to a variable after using it in an expression.
  - The pre-decrement operator subtracts 1 to a variable before using it in an expression.
  - The post-decrement operator subtracts 1 to a variable after using it in an expression.
    - E.g.:

```
int a = 0;
printf("%d\n", a);
printf("%d\n", a++);
printf("%d\n", a);
printf("%d\n", --a);
printf("%d\n", a);
```



- The increment and decrement operators have the same level of precedence as the **not** operator.

Numerical Conversion Specifiers in *printf* and *scanf*

• **Type**	**printf**	**scanf**
• short	`%d`, `%i`, `%hd`, `%hi`	`%hd`, `%hi`
• int	`%d`, `%i`	`%d`, `%i`
• long	`%ld`, `%li`	`%ld`, `%li`
• unsigned short	`%hu`	`%hu`
• unsigned int	`%u`	`%u`
• unsigned long	`%lu`	`%lu`
• float	`%f`, `%e`, `%E`, `%g`, `%G`	`%f`, `%e`, `%E`, `%g`, `%G`
• double	`%f`, `%e`, `%E`, `%g`, `%G`	`%lf`, `%le`, `%lE`, `%lg`, `%lG`
• long double	`%Lf`, `%Le`, `%LE`, `%Lg`, `%LG`	`%Lf`, `%Le`, `%LE`, `%Lg`, `%LG`

- Careful with the conversion specifier for a `double` in a `printf`.
- The conversion specifier `%e` prints out the number in scientific notation: `1.0e0`
- The conversion specifier `%E` prints out the number in scientific notation: `1.0E0`
- The conversion specifier `%g` prints out the number in `%f` or in `%e`
- The conversion specifier `%G` prints out the number in `%f` or in `%E`

Case Study – Temperature Conversion

Problem: design a software that prompts the user to enter a temperature in Fahrenheit and then outputs its equivalent in Celsius.

- Using the problem solving methodology
  - Step 1: Problem Identification and Definition
  - Step 2: Gathering of Information and I/O Description
  - Step 3: Algorithm Development and Verification
  - Step 4: Pseudo code
  - Step 5: Implementation
  - Step 6: Software Testing and Validation

Step 1: Problem Identification and Definition

- A software shall be developed to convert the temperature entered by the user from Fahrenheit to Centigrade.

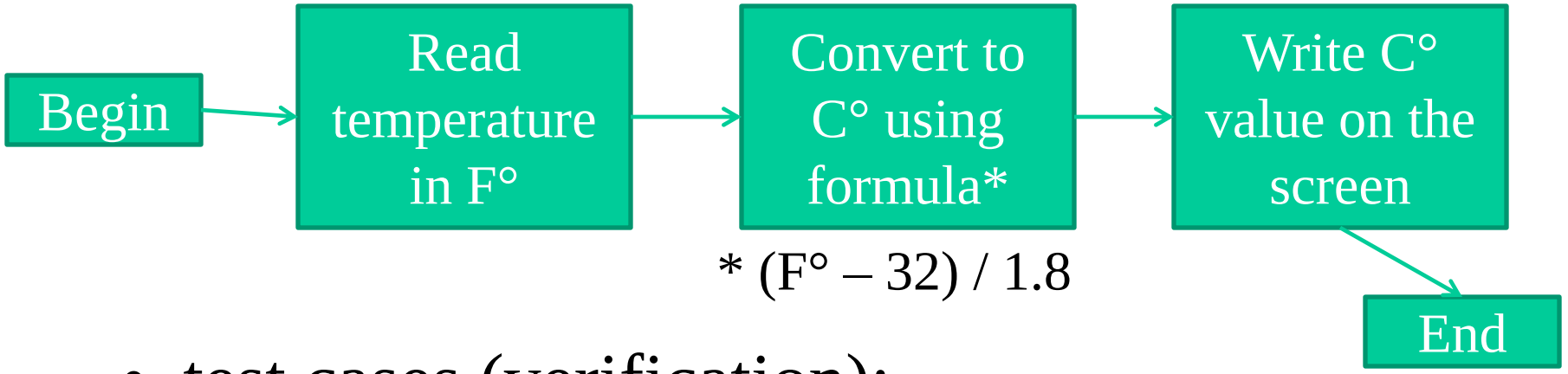
Step 2: Gathering of Information, I/O Description

- Equation to make the conversion

$$C° = (F° – 32) / 1.8$$

- Input/output description

Step 3: Algorithm Development and Verification



- test cases (verification):
  - Test case 1:
    - F = 0: C = (0 – 32) / 1.8 = – 17.76
  - Test case 2:
    - F =32: C = (32 – 32) / 1.8 = 0
  - Test case 3:
    - F = 193: C = (193 – 32) / 1.8 = 89.45

Step 4: Pseudo Code

Print "Please enter the temperature in Fahrenheit"

Read a value into F_Temp

Assign (F_Temp - 32)/1.8 to C_Temp

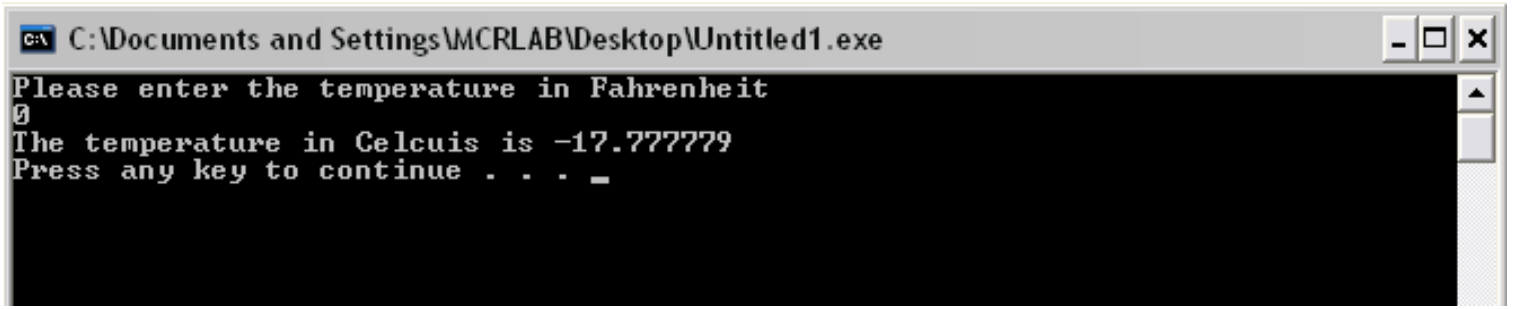
Print "The temperature in Celcuis is", C_temp

Step 5: Implementation

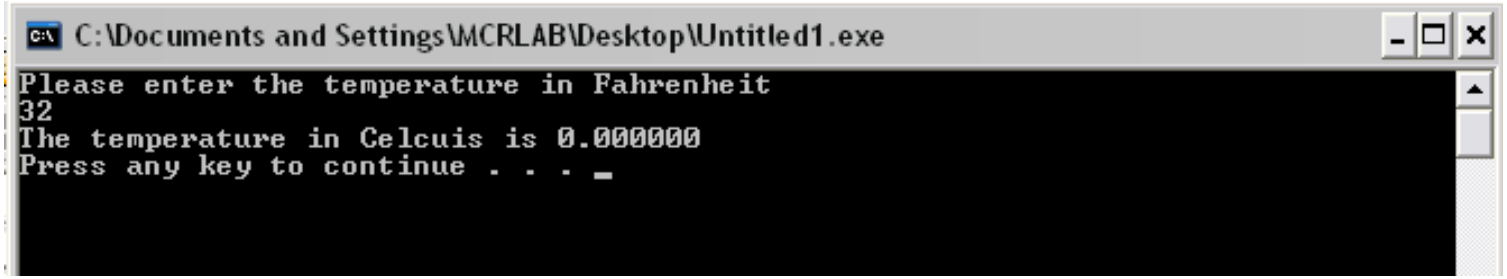
```
#include <stdio.h>

// main function
void main()
{
    float F_Temp, C_Temp;

    printf("Please enter the temperature in Fahrenheit\n");
    scanf("%f", &F_Temp);
    C_Temp = (F_Temp - 32)/1.8;
    printf("The temperature in Celcius is %f\n", C_Temp);
    system("PAUSE");
}
```

Step 6: Testing and Validation

- Test case 1:



- Test case 2:



- Test case 3: