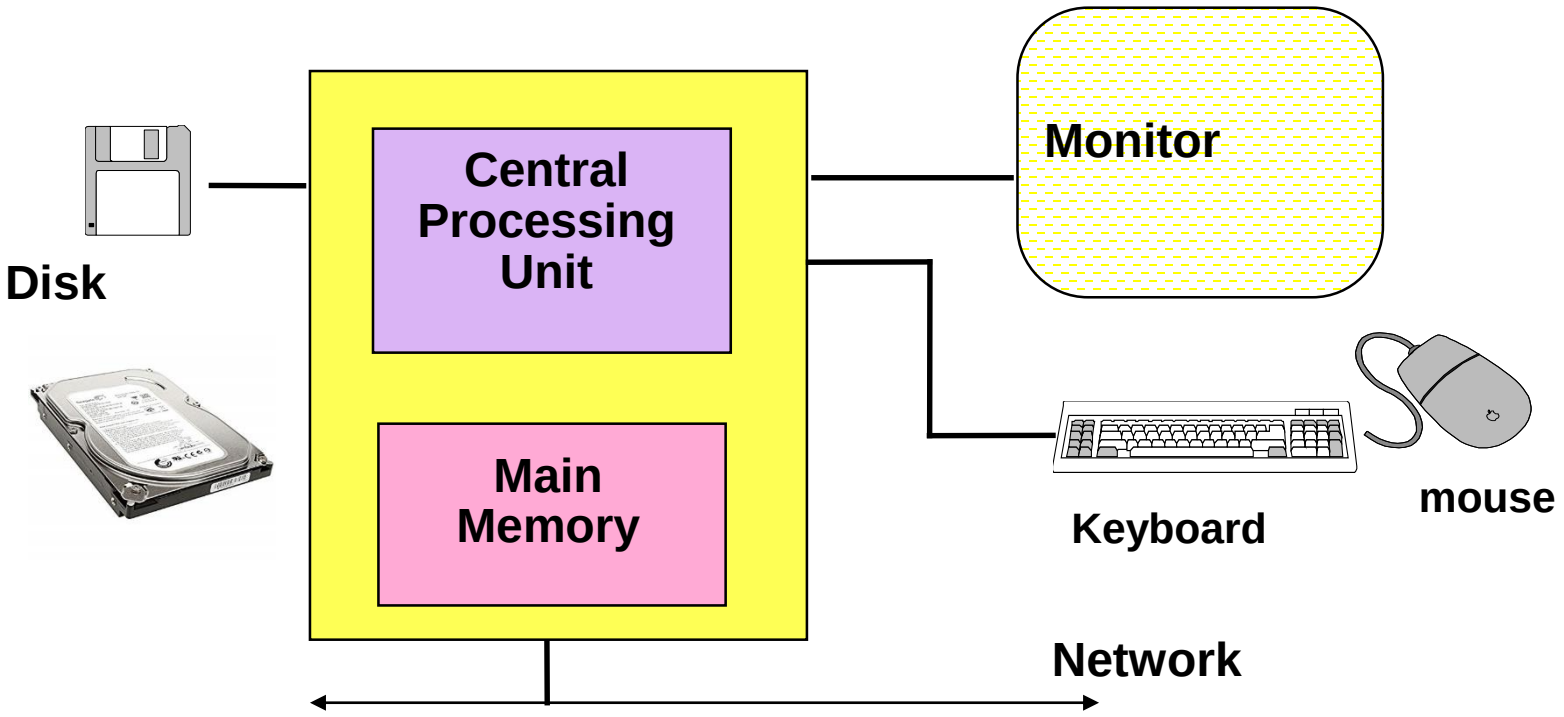
Variables, Values and Types

Session 2

Overview

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

Review: Computer Organization



Review: Memory

Memory is a collection of locations called variables

In a programming language, we get at the location by using a variable

Each variable has

- A name (an identifier)
- A type (the kind of information it can contain)

Variables

- Refer to memory location where a particular value is stored
- Type of data decides the amount of memory allocated to variables
- Names assigned to variables to store a particular data, help us in retrieving the data as and when required

Memory and Variables

Memory is a collection of locations called variables

In a programming language, we get at the location by using a variable

Each variable has

- A name (an identifier)
- A type (the kind of information it can contain)

Basic types include

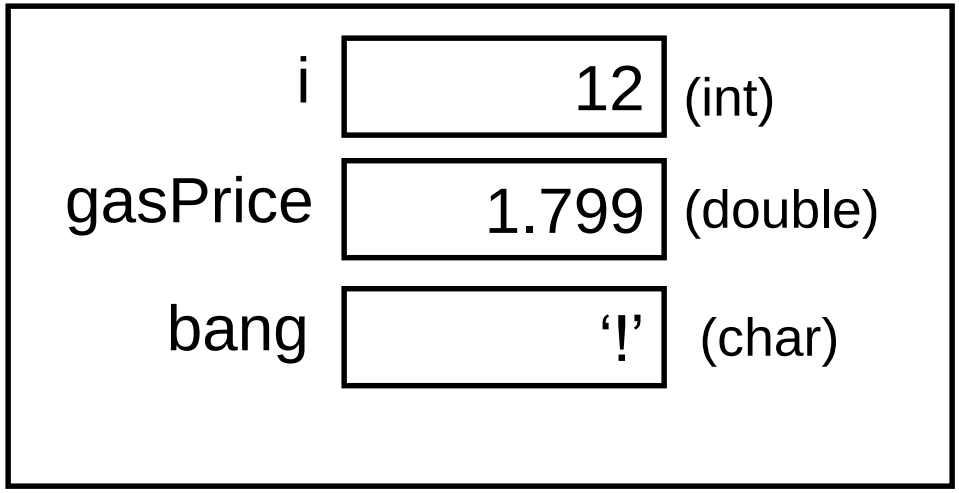
- int (integers – whole numbers: 17, -42)
- double (floating-point numbers with optional fraction and/or exponent: 3.14159, 6.02e23)
- char (character data: ‘a’, ‘?’, ‘N’, ‘ ’, ‘9’)
  - Note: ‘9’ is a character; 9 is an integer – they are different and have different types

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"

Data Type

- Decides the amount of memory to be allocated to a variable to store a particular type of data
- Declaring a variable
  - Allocates memory
  - Portion of memory is referred to by the variable name
- General form of declaring a variable
  - **data type** (variable name)
- Common data types
  - Numeric
  - Alphanumeric

Data Types

- **Type int**
  - Stores numeric data
  - Consists of a sequence of one or more digits
  - Includes only whole numbers
- **Type char**
  - Stores a single character
  - The single character is enclosed within two single quotation marks
  - Digits can also be stored as characters but cannot be used for calculations

Data Types (Contd.)

- **Type float**
  - Stores values containing decimal places
  - Stores either whole or fractional numbers
- **Type double**
  - Stores twice the number of digits than a **float** type
  - Occupies double memory space than a **float**

- Precise number of digits stored by **float** and **double** types depends upon the particular computer system

Derived Data Types

- Modifiers used to alter the meaning of the data type to fit various situations more precisely
- **unsigned**
  - Specifies that a variable can take only positive values
  - Used with the **int** and **float** data types by prefixing it with the word **unsigned**
- **long** and **short**
  - Used when an integer of longer or shorter length than the usual length is required
  - A long integer is written as **long int** or just **long**
  - A short integer is written as **short int** or **short**

Data Types (Contd.)

The various data types and their memory requirements

Table 3-1. C# built-in value types

Type	Size (in bytes)	.NET Type	Description
byte	1	Byte	Unsigned (values 0-255).
char	1	Char	Unicode characters.
bool	1	Boolean	true or false.
sbyte	1	Sbyte	Signed (values -128 to 127).
short	2	Int16	Signed (short) (values -32,768 to 32,767).
ushort	2	Uint16	Unsigned (short) (values 0 to 65,535).
int	4	Int32	Signed integer values between -2,147,483,647 and 2,147,483,647.
uint	4	Uint32	Unsigned integer values between 0 and 4,294,967,295.
float	4	Single	Floating point number. Holds the values from approximately +/-1.5 * 10^{-45} to approximate +/-3.4 * 10^{38} with 7 significant figures.
double	8	Double	Double-precision floating point; holds the values from approximately +/-5.0 * 10^{-324} to approximate +/-1.7 * 10^{308} with 15-16 significant figures.
decimal	8	Decimal	Fixed-precision up to 28 digits and the position of the decimal point. This is typically used in financial calculations. Requires the suffix "m" or "M."
long	8	Int64	Signed integers ranging from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807.
ulong	8	Uint64	Unsigned integers ranging from 0 to 0xffffffffffffffff.

Variables (Examples)

- The area of a rectangle is given by:
  - Area = A = Length x Breadth = L x B

- The simple interest is given by:
  - Interest = I = Principal x Time x Rate / 100 = P x T x R / 100

Variables (Example)

- The sum of the marks obtained by five students is calculated as follows:
  - Sum = 24 + 56 + 72 + 36 + 82
- To calculate the average of the marks, the variable sum can be used as follows:
  - Avg = Sum / 5

Guidelines for specifying Variable Names

- Must begin with an alphabet
- First character to be followed by a sequence of letters or digits or special character ‘underscore’
- Avoid using letter O in situations where it can be confused with the number 0 and the lowercase letter l can be mistaken with the number 1
- Uppercase and lowercase letters are treated different
- Name of the variable should be descriptive of the value it holds

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 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!

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

An Example

```
/* calculate and print area of 10x3 rectangle */
void Main(string [] args)
{
        int     rectangleLength;
        int     rectangleWidth;
        int     rectangleArea;
        rectangleLength = 10;
        rectangleWidth = 3;
        rectangleArea = rectangleLength * rectangleWidth ;

        Console.WriteLine("with length {0} and width {1} the area of rectangle is {2}",
rectangleLength, rectangleWidth ,rectangleArea);


}
```

Hand Simulation

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

Tracing the Program

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

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.

Declaring vs Initializing

```
void Main(string [] args)
{
      double income;                /*declaration of income, not an
                                        assignment or initialization */
      income = 35500.00;              /*assignment to income,
                                          initialization of income, not a declaration.*/
Console.WriteLine("Old income is {0}", income);
      income = 39000.00;               /*assignment to income, not a
                                       declaration,or initialization */
Console.WriteLine("After raise: {0}", income);

}
```

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

```
void Main(string [] args)
{
    double fahrenheit, celsius;

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

}
```

Fahrenheit to Celsius (II)

```
void Main(string [] args)
{

    double fahrenheit, celsius;
    Console.WriteLine("Enter a Fahrenheit temperature: ");
    fahrenheit=Convert.ToDouble(Console.ReadLine());
    celsius =  (fahrenheit - 32.0) * 5.0 / 9.0;
    Console.WriteLine("That equals {0} degrees Celsius.", celsius);
}
```

Running the Program

Enter a Fahrenheit temperature: 45.5
That equals 7.500000 degrees Celsius

Program trace

	fahrenheit	***celsius***
after declaration	**?**	**?**
after first *WriteLine*	**?**	**?**
after *ReadLine*	**45.5**	**?**
after assignment	**45.5**	**7.5**
after second *WriteLine*	**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)

```
void Main(string [] args)
{
      double fahrenheit, celsius;
      Console.WriteLine("Enter a Fahrenheit temperature: ");
      Fahrenheit = Convert.ToDouble(Console.ReadLine());
      celsius = fahrenheit - 32.0 ;
      celsius = celsius * 5.0 / 9.0 ;
      Console.WriteLine("That equals {0} degrees Celsius.",  celsius);
}
```

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!

Convert class in C#

- Converts a base data type to another base data type.
- Common methods
  - Convert.ToInt16
  - Convert.ToInt32
  - Convert.ToDouble
  - Convert.ToBoolean
  - Convert.ToChar
  - Convert.ToDateTime