Variables (1)

Storing and manipulating values
Goals

- By the end of this lesson you will be able to:
  - Create variables and store values in them
  - Get input from the keyboard and store it in a variable
  - Output variables and expressions to the terminal
  - Understand how expressions are evaluated
  - Understand the rules and conventions for variable identifiers
  - Understand the order of operations and operator precedence for simple mathematical operations in C++
Creating variables

- Recall that the basic data types in C++ are:
  - `int` – integers
  - `double` – real numbers
  - `char` – characters
  - `bool` – Boolean truth/false value

- We would like to get input from the user, store it in a variable, then work with it to produce some meaningful output
Identifier rules

• Identifiers may only consist sequences of:
  - Alphabetic characters (e.g. A, g, z, E)
  - Digits (e.g. 0, 5, 8)
  - The underscore _

• Identifiers cannot:
  - Contain any special characters or spaces
  - Begin with a digit
  - Be a reserved word
    • A reserved word is one built into the language such as `int`, `char`, or `long` – you will learn more reserved words as you learn more about C++
  - Be identical to other identifiers in the same scope
Identifier conventions

- Identifiers should not start with an underscore _ as these are typically used for header file identifiers.

- Variable identifiers should not be all capital letters as these are typically used for named constants which we shall learn about very soon.

- Identifiers should be descriptive about what is to be stored in them (e.g. `overdraft_fee` is better than `odf`).

- Some additional naming conventions exist such as Hungarian notation and CamelCase.
Declaring variables

• Variables can be declared using the following syntax:
  
  datatype identifier;
  
• For example:
  
  int age;
  char c;
  double balance;
  
• Multiple variables of the same type can be declared in one line if their identifiers are separated by a comma:
  
  double balance, interest_rate, fee;
  
• Variables need to be declared before you do anything with them

• Variables should be declared inside a function (e.g. main) and immediately before you need to use them
Declaring variables

- You should only declare variables that you will both need and use.
- It is bad style to reuse a variable used with a particular identifier for something unrelated to that identifier.
- For example in a program where:
  - The user enters their age
  - You output their age
  - The user enters their birth year
  - You output their birth year
- You should use separate variables for age and birth year.
Basic output

- We’ll want to display the variables that we create and manipulate, so let’s look at cout syntax in more detail:

  cout << var_or_expr << var_or_expr << ... ;

- Variables or expressions we’ll be using for now include:
  - One of our variables (e.g. fee)
  - A mathematical expression (e.g. 5 + 5 or fee + 3)
  - A character (e.g. ' $ ')
  - A string of characters(e.g. "Hello world")
  - endl (to put a newline on the output)

- You can output as many things as you like, so long as each is separated by the stream insertion operator <<=
Basic input

- To get input into a variable we can use `cin` (common input)
  ```
  cin >> var;
  ```
- Variables that are input are separated by whitespace
  ```
  int age;
  cout << "Enter your age: ";
  cin >> age;
  ```
- You can get input for multiple variables at the same time, so long as each is separated by the *stream extraction operator* `>>`
  ```
  cin >> x >> y >> z;  // if user types in 3 8 15 this will work as expected
  ```
- The user must put white space between each value
- For style reasons:
  - When prompting for input you will not want to put `endl` at the end of the prior `cout`
  - Prompt for each variable separately
Assigning value

- You can assign value to a variable using the assignment symbol =
  
  ```
  int x, y;
  
  x = 5;
  y = 20;
  x = y;  // x is now 20
  x = 3;   // x is now 3, y is still 20
  ```

- You can assign values to a variable when you declare it – this is called initialization
  
  ```
  int x, y = 15, z;
  bool has_overtime = true;
  char middle_initial = 'Z';
  ```

- The left-side of the assignment must be a variable
  
  ```
  2 + z = 5; is not allowed
  ```
Basic mathematical operators

- There are five basic mathematical operators in C++:
  - + addition
  - – subtraction
  - * multiplication
  - / division
  - % modulo or mod – this is the remainder when one number is divided by another (e.g. $16 \% 5 = 16 \text{ mod } 5 = 1$)

- These operators are examples of **binary operators** – this is not in reference to the binary numeral system, but based on the fact that the operators take two operands (e.g. $5 + 3, 8.8 \times 4$)
Expressions & order of operations

- **Expressions** in C++ are a computation consisting of a sequence of operators and operands and are **evaluated**
  - \[ 3 + 4 + 5 \] is an expression
  - \[ 3 + 4 + 5 \] evaluates to 12
- The five mathematical operators are evaluated left-to-right with the following precedence:
  - * / % (higher precedence)
  - + - (lower precedence)
- \[ 3 + 5 \times 5 \] will evaluate to 28, not 40
• Parentheses can be added around any expression and it will be evaluated before the rest of the expression:
  \[(3 + 5) \times 5\] will evaluate to 40
  \[((3 + 4)) + ((5))\] looks strange/bad but is legal

• + and – are also **unary** (one argument) operators – C++ knows how to interpret what you mean
  \[+3 --5\] will evaluate to 8

• Spaces between operands and operators are not required, but are a component of good style: \[+3--5\] doesn’t look good

• = is a binary operator as well, but it has a lower precedence than everything we have encountered
• Now a more complete picture of operator precedence can be given as follows:

- ( ) parentheses (highest precedence)
- - + unary
- * / %
- + - binary
- = (lowest precedence)
A complicated example...

- Note that while = is assignment, the result of the assignment operation is an evaluation to the variable on the left side of that assignment.

- The order of evaluation $x = 2 \times (y = z = 6 - 8 \mod 3 + 2)$ is well defined:
  - $8 \mod 3$ is evaluated to $2$
  - $6 - 2$ is evaluated to $4$
  - $4 + 2$ is evaluated to $6$
  - $z$ is assigned the value of $6$
  - $y$ is assigned the value of $z$, which is $6$
  - $x$ is assigned the value of $2 \times y$, which is $12$

- If you `cout` this expression, the output will be $12$ (the value of $x$)
Exploration

• Write a program that gathers input from the user and performs some calculations

• Example programs:
  – Find the area of a circle given its radius
  – Find the area of a rectangle given its length and width
  – Given the number of hours and minutes, calculate the number of seconds