C++ Basics

Chapter 8
Objectives

In this chapter, you will learn about C++:

- Variables and Assignments
- Input and Output Statements
- Data Types and Expressions
- Simple Flow of Control
- Program Style
C++ Variables

- Basic Concept
  - Variables Identify Memory locations/cells
  - Variables provide a way to store and access information
    - Your Program can write a value in them
    - Your Program can change the value stored in a Variable
Creating Variables

Identifiers

- We identify Variables by Names (Identifiers)
- Rules for Choosing Identifiers
  - First character must be
    - A Letter
    - Underscore character
  - Trailing characters must be
    - Letters
    - Numbers
    - Underscore character
  - Can be of any length
Creating Variables

Identifiers

- C++ is Case-Sensitive
  - Distinguish between Upper and Lower case
  - Four Distinct Variables (memory locations):
    RATE Rate RATe rate

Always use meaningful and descriptive names for variables
Keywords
Special Identifiers

- Keywords are reserved for C++ language:
  - int, double, float
  - Cannot be used as Identifiers

Are cin and cout keywords?
Variable Declarations

Type_Name Variable_Name_1, variable_Name_2, …

- A Variable must be declared before they can be used (referenced)
  - Why? To Tell the Compiler what type of data to store in it – Why?
    - Different type of variables require different size of memory location

Examples of Variable Declarations

```plaintext
int number_of_bars;  // Stores a whole number – e.g., -3, -2, 0, 1, 2, 3...

double rent, tax;  // Each stores a fractional number – e.g., -3.12, -0.2, 0.0, 0.1, 2.75...
```
Variable Declaration
Statement Location

■ Declare Variables at:

- **Beginning of Program**

```cpp
int number_of_pods, peas_per_pod, total_peas;
cout << “Enter the number of pods:\n”; cin >> number_of_pods;
cout << “Enter number of peas in a pod:\n”; cin>> peas_per_pod;

total_peas = number_of_pods * peas_per_pod;
```

- **Immediately Prior to use**

```cpp
int number_of_pods, peas_per_pod;
cout << “Enter the number of pods:\n”; cin >> number_of_pods;
cout << “Enter number of peas in a pod:\n”; cin>> peas_per_pod;

int total_peas;

total_peas = number_of_pods * peas_per_pod;
```

Did you notice the semicolon at the end of the declaration statement?
# Variable Types

<table>
<thead>
<tr>
<th>TYPE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>Integers</td>
</tr>
<tr>
<td>double</td>
<td>Real numbers (2.15, -0.25,..)</td>
</tr>
<tr>
<td>char</td>
<td>Characters (holds any single character): ‘A’, ‘%’</td>
</tr>
<tr>
<td></td>
<td>But “A” is a string not a char</td>
</tr>
<tr>
<td>bool</td>
<td>Boolean Expression: true or false</td>
</tr>
<tr>
<td>String</td>
<td>need: #include &lt;string&gt; and using namespace std;</td>
</tr>
</tbody>
</table>
Assignment Statements

- Assignment Statement:
  - Sets the value of a variable

- Syntax
  - Variable = Expression;

number_of_students = 32;
number_of_cs_majors = 25;
new_students = 4;
number_of_minors = number_of_students – number_of_majors;
number_of_students = number_of_students + 4;

Initialization
## Expressions

### Arithmetic

<table>
<thead>
<tr>
<th>Formula</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b^2 - 4ac$</td>
<td>$b<em>b - 4</em>a*c$</td>
</tr>
<tr>
<td>$x(y + z)$</td>
<td>$x*(y + z)$</td>
</tr>
<tr>
<td>$\frac{1}{x^2 + x + 3}$</td>
<td>$1/(x*x + x + 3)$</td>
</tr>
<tr>
<td>$X^2 + X + 3$</td>
<td></td>
</tr>
</tbody>
</table>
Variables

Initialization

- Sets variable to its initial value
- Method
  - Initialize Variable After Declaration
    ```
    int number_of_students;
    number_of_students = 32;
    ```
  - Initialize Variable in Declaration
    ```
    int number_of_students = 32;
    ```

Why do we initialize variables?
Input and Output Streams

- **Data Stream** – Sequence of Data Stream
- **Input** data Stream
  - Sequence of Data your Program Uses (reads)
  - May originate from keyboard, file etc…
- **Output** data Stream
  - Sequence of Data your Program Generates (output)
  - Destination may be Monitor, file
Output Stream

\texttt{cout}

- \texttt{cout} is an output stream to send data from your program to monitor
  - \texttt{cout \textless\textgreater number\_of\_bars \textless\textgreater " candy bars\n"};

\begin{itemize}
  \item Insertion operator sends the value of \texttt{number\_of\_bars} to \texttt{cout}
  \item Next, the Insertion operator sends the string “ candy bars” followed by a new line character to the \texttt{cout} stream
\end{itemize}

\textbf{Did you notice}
\begin{itemize}
  \item 1. No spaces allowed between the two symbols <
  \item 2. Only Double Quotes are allowed for inserting strings in the stream
\end{itemize}
Two cout statements instead of one

```cpp
cout << number_of_bars;
cout << " candy bars\n";
```

Arithmetic Expressions in cout

```cpp
cout << "Total cost is $" << (price + tax);
```

Suppose cout statement in your program exceeds length of screen:

```cpp
cout << number_of_bars<< " candy bars\n"<< one_weight
<< " ounces each\n";
```
Use Include Directives to add Libraries to your program (e.g., iostream library)
#include<iostream>
- The iostream library contains cout and cin
- Gives your program access to cout and cin

To use names, such as cin and cout defined in std library in your program
- using namespace.std
# Escape Sequence

Allows you to put special characters into a string:

<table>
<thead>
<tr>
<th>Sequence</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>\</code></td>
<td>Prints one backslash</td>
</tr>
<tr>
<td><code>&quot;</code></td>
<td>Prints a double quote</td>
</tr>
<tr>
<td><code>\a</code></td>
<td>Sounds a system bell (alert)</td>
</tr>
<tr>
<td><code>\n</code></td>
<td>Move cursor to beginning of next line</td>
</tr>
<tr>
<td><code>\t</code></td>
<td>Horizontal tab</td>
</tr>
</tbody>
</table>

How do you include backslash, quote, alert or tab character in cout statement?

*The compiler interprets the first backslash as the start of an escape sequence*
Escape Sequence
Blank Line

- To Insert Blank Line in cout:
  - `cout << "\n";`
  - Or
  - `cout << endl;`
### Real Numbers

#### Output Format

- **Why Format Real Numbers?**
- **Desired Output Format is unpredictable**

```cpp
rent = 422.5;
cout << "Rent is $" << rent << endl;
```

#### Possible Outputs:

- Rent is $422.5
- Rent is $422.5000
- Rent is $422.50
- Rent is $4.225000e02
- Rent is $422.50
Real Numbers
Output Format

- Insert before the first `cout`:
  
  ```cpp
  setf(ios::fixed);
  setf(ios::showpoint);
  precision(n);
  ```

  **Specify Fixed point notation**

  **Show decimal point**

  **Specify n decimal places**

  Where `n` is either:
  - Integer value: 0, 1, 2, 3, … or
  - Initialized Integer Variable

  ```cpp
  int n = 2;
  ```

  **Example**

  ```cpp
  setf(ios::fixed);
  setf(ios::showpoint);
  precision(2);
  rent = 422.5;
  cout << “Rent is $” << rent << endl;
  ```

  Rent is $422.50

  **What if you wish to change subsequent decimal places for cout?**
Input Streams

cin

- cin (see-in)
  - Extracts data entered from (keyboard) to a variable

Example

```cpp
cout << "Enter the number of pods:\n" << "Enter number of peas in a pod:\n";
cin >> number_of_pods;
cin >> peas_per_pod;
```

Dialogue

Enter number of pods:
Enter number of peas in a pod:
4 10 [return]

Delimiter: Blank space

What happens if user enters one input or more than two inputs?
Input Streams

**cin**

- Always prompt user for the desired number (and type) of inputs
- Echo input read using `cout` statement

```cpp
cout << "Enter the number of pods:\n";
    << "Enter number of peas in a pod:\n";
cin >> number_of_pods;
cout << number_of_pods << " was entered.\n";
cin >> peas_per_pod;
cout << peas_per_pod << " was entered." endl;
```
Control Structures – Basic Labs

- **Sequential Execution**
  - Statements are executed serially

- **Branch**
  - Decide between two (or more) choice of statements
    - Need a criterion (or condition)

- **Iterative**
  - Repeat execution for a set of statements
Branch Construct

Single Statement

if (condition)

**True statement**

else

**False statement**

Boolean Expression: True or False

The *True statement* is executed if condition is True

Otherwise the *False statement* is executed

Example:

```plaintext
if (hours > 40)
gross_pay = rate*40 + 1.5*rate*(hours - 40);
else
gross_pay = rate*hours;
```

*Did you notice the mandatory parenthesis in the if statement?*
Branch Construct
Block of Statements

if (condition)
{
    statement_1
    statement_2
    statement_n
}
else
{
    statement_1
    statement_2
    statement_n
}

Boolean Expression: True or False

The True statements are executed if condition is True
Otherwise the False statements are executed

Did you notice the mandatory braces for each block of statements
## Boolean Expressions

### Comparison Operators

<table>
<thead>
<tr>
<th>Logical Operator</th>
<th>C++ Example Comparison</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>==</td>
<td>(x + 7) == 2*y</td>
<td>equal to</td>
</tr>
<tr>
<td>!=</td>
<td>ans != n</td>
<td>not equal to</td>
</tr>
<tr>
<td>&lt;</td>
<td>count &lt; (m+ 3)</td>
<td>less than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>time &lt;= limit</td>
<td>less than or equal to</td>
</tr>
<tr>
<td>&gt;</td>
<td>time &gt; limit</td>
<td>greater than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>age &gt;= 21</td>
<td>greater than</td>
</tr>
</tbody>
</table>
Boolean Expressions

&& (AND) Operator

- && Operator
  - Combines two Boolean expressions
  - Compound expression is true if both expressions are true
  - Let’s review TRUTH Table for “AND” operator

Syntax:

\[(\text{Boolean Expression}_1) \text{ && } (\text{Boolean Expression}_2)\]

Example: if \((x > 4) \text{ && } (x < 10)\)  
True for what values in variable x?

Did you notice the mandatory parenthesis in blue fonts?
Boolean Expressions
|| ("or") Operator

- || Operator
  - Combines two Boolean expressions
  - Compound expression is False if both expressions are false
  - Otherwise it is true
  - Let’s review TRUTH Table for “or” operator

Syntax:

```
(Boolean Expression_1) || (Boolean Expression_2)
```

Example

```
if ( (x ==1) || ( x > 4) )
```

True for what values in variable x?

Did you notice the mandatory parenthesis in blue fonts?
Boolean Expressions

! (“not”) Operator

Use ! ( ...) to Negate a Boolean expression

- Let’s review TRUTH Table for “not” operator

Syntax: ! (Boolean Expression)

Examples

if (! (x ==1))

if (! (! ( x ==1) || ( x > 4) ))

Did you notice the mandatory parenthesis in blue fonts?
Boolean Expressions
Compound statements

- Assume variables: rent, grocery, phone_bill
- Write the C++ Boolean expression to compare:

  phone_bill < grocery < rent
While Loops

- **Repeat** one or more operations:
- **Example:**
  ```cpp
  int count_down = 4;
  while (count_down > 0)
  {
      cout << "Hello ";
      count_down = count_down -1;
  }
  cout << endl;
  ```

  Output:
  
  ```plaintext
  Hello Hello Hello Hello Hello
  ```
### While Loop Operations

```cpp
int count_down = 4;
while (count_down > 0) {
    cout << "Hello ";
    count_down = count_down - 1;
}
```

1. Evaluate condition
2. If condition is False, **skip** loop body to `cout << endl;`
3. If condition is True,
   - Execute loop body
   - Go back to step 1

**Did you notice Boolean expression is always changed in the loop body?**
While Loop
Syntax

Multiple Statements:

while (Boolean Expression)
{
    Statement_1
    Statement-2
    Statement_n
}

One Statement:

while (Boolean Expression)

Where do you put the semi-colon(s)?
While Loop
do – while statement

- Use only if loop body should be executed at least once
- Multiple Statements:

```java
do {
    Statement_1
    Statement_2
    ------
    Statement_n
} while (Boolean Expression);
```

The Boolean expression is checked after the body is executed

Did you notice the mandatory semicolon
While Loop

do – while statement

- Use only if loop body should be executed at least once
- Single Statement:

\[
\text{do} \quad \text{Statement} \quad \text{while (Boolean Expression)};
\]

The Boolean expression is checked after the single statement is executed.

Did you notice the mandatory semicolon
Constants

- Number constants have no mnemonic value
- Number constants used throughout a program are difficult to find and change when needed

Constants
- Allow us to name number constants so they have meaning
- Allow us to change all occurrences simply by changing the value of the constant
**Constant Declaration**

`const`

- `const` is the keyword to declare a constant
- Example:
  ```
  const int WINDOW_COUNT = 10;
  ```
  declares a constant named `WINDOW_COUNT`

  - Its value cannot be changed by the program like a variable
  - *It is common to name constants with all capitals*
```cpp
//
#include <iostream>
using namespace std;
int main( )
{
    int number_of_bars;
    double one_weight, total_weight;
    cout << "Enter the number of candy bars in a package\n";
    cout << "and the weight in ounces of one candy bar.\n";
    cout << "Then press return.\n";
    cin >> number_of_bars;
    cin >> one_weight;
    total_weight = one_weight * number_of_bars;
    cout << number_of_bars << " candy bars\n";
    cout << one_weight << " ounces each\n";
    cout << "Total weight is " << total_weight << " ounces.\n";
    cout << "Try another brand.\n";
    cout << "Enter the number of candy bars in a package\n";
    cout << "and the weight in ounces of one candy bar.\n";
    cout << "Then press return.\n";
    cin >> number_of_bars;
    cin >> one_weight;
    total_weight = one_weight * number_of_bars;
     cout << number_of_bars << " candy bars\n";
    cout << one_weight << " ounces each\n";
    cout << "Total weight is " << total_weight << " ounces.\n";
    cout << "Perhaps an apple would be healthier.\n";
    return 0;
}
```
C++ Sample Program

if-else statement

```cpp
// if-else Statement
#include <iostream>
using namespace std;
int main() {
    int hours;
    double gross_pay, rate;

    cout << "Enter the hourly rate of pay: $";
    cin >> rate;
    cout << "Enter the number of hours worked, \n"
         << "rounded to a whole number of hours: ";
    cin >> hours;

    if (hours > 40)
        gross_pay = rate*40 + 1.5*rate*(hours - 40);
    else
        gross_pay = rate*hours;

    cout.setf(ios::fixed);
    cout.setf(ios::showpoint);
    cout.precision(2);
    cout << "Hours = " << hours << endl;
    cout << "Hourly pay rate = $" << rate << endl;
    cout << "Gross pay = $" << gross_pay << endl;
    return 0;
}
```