# Chapter 3

## SOME MORE BASICS

# Outline

- Assignment statement
- Program input using the *cin* object
- Formatting the output
- Using mathematical library functions
- Type Conversions
- Octal and hexadecimal number
- Block statement

# Overview

- **In the last chapter, we studied how result are displayed and how numerical data are stored and processed using variables and assignment statements.**

- **In this chapter, we study some C++'s additional processing and input capabilities.**

# Assignment

- **How do we place data items into variables?**
  - Read in values typed at the keyboard by the user
  - Use an assignment statement

- **Assignment statement examples:**

```
length = 25;
cMyCar = “Mercedes”;
sum = 3 + 7;
newtotal = 18.3*amount;
slope = (y2 – y1)/(x2 – x1);
```

# Assignment statement

- *Assignment operator* (=) are used for assignment of a value to a variable and for performing computations.

- Assignment statement has the syntax:

  *variable = expression*;

- *Expression* is any combination of constants, variables, and function calls that can be evaluated to yield a result.

# Assignment statement (cont.)

- **The order of events when the computer executes an assignment statement is**

  - Evaluate the expression on the right hand side of the assignment operator.

  - Store the resultant value of the expression to the variable on the left hand side of the assignment operator.

- **Note:**

  1. The equal sign here does not have the same meaning as an equal sign in mathematics.

  2. Each time a new value is stored in a variable, the old one is overwritten.

# Assignment Variations

- **C++ includes other arithmetic operators in addition to the equal sign.**

| Operator | Example | Meaning |
|---|---|---|
| = | iNum1 = iNum2 | |
| += | iNum1 += iNum2 | iNum1 = iNum1 + iNum2 |
| -= | iNum1 -= iNum2 | iNum1 = iNum1 - iNum2 |
| *= | iNum1 *= iNum2 | iNum1 = iNum1 * iNum2 |
| /= | iNum1 /= iNum2 | iNum1 = iNum1 / iNum2 |
| %= | iNum1 %= iNum2 | iNum1 = iNum1 % iNum2 |

- ***sum* += *10* is equivalent to *sum* = *sum* + *10***

# Increment and decrement operators

- **For the special case in which a variable is either increased or decreased by 1, C++ provides two unary operators: *increment operator* and *decrement operator*.**

| Operator | Description |
|---|---|
| ++ | Increase an operand by a value of one |
| - - | Decrease an operand by a value of one |

- **The *increment* (++) and *decrement* (--) unary operators can be used as prefix or postfix operators to increase or decrease value.**

# Increment and decrement operators (cont.)

- **A *prefix* operator is placed *before* a variable and returns the value of the operand *after* the operation is performed.**

- **A *postfix* operator is placed *after* a variable and returns the value of the operand *before* the operation is performed.**

- **Prefix and postfix operators have different effects when used in a statement**

b = ++a;  // prefix way

will first increase the value of a to 6, and then assign that new value to b. It is equivalent to

a = a +1; b = a;

b = a++; // postfix way

will first assign the value of 5 to *b*, and then increase the value of *a* to 6. It is equivalent to

b = a; a = a + 1;

- **The decrement operators are used in a similar way.**

b = --a;

equivalent to

a = a -1; b = a;

b = a--;

equivalent to

b = a; a = a -1;

# Exercise on class

- What are the results of the following program?

```
#include <iostream.h>
int main()
{
   int c;
   c = 5;
   cout << c << endl;
   cout << c++ << endl
   cout << c << endl << endl;
   c = 5;
   cout << c << endl;
   cout << ++c << endl;
   cout << c << endl;
   return 0;
}
```

# PROGRAM INPUT USING THE cin OBJECT

- So far, our programs have been limited since that all their data must be defined within the program source code.
- We now learn how to write programs which enable data to be entered via the keyboard, while the program is running.
- **Standard Input Stream**
  - The *cin* object reads in information from the keyboard via the standard input stream.
  - The *extraction operator* (>>) retrieves information from the input stream.
  - When the statement *cin >> num1;* is encountered, the computer stops program execution and accepts data from the keyboard. When a data item is typed, the *cin* object stores the item into the variable listed after the >> operator.

**Example 3.4.1**

```
#include <iostream.h>
int main()
{
  float num1, num2, product;
  cout << "Please type in a number: ";
  cin  >> num1;
  cout << "Please type in another number: ";
  cin  >> num2;
  product = num1 * num2;
  cout << num1 << " times " << num2 << " is " << product << endl;
  return 0;
  }
```

**The output of the above program**:

Please type in a number: 30

Please type in another number: 0.05

30 times 0.05 is 1.5

# Exercise on class

**Write a program that gets 3 numbers and prints out their average**

```
#include <iostream.h>
int main()
{
  int num1, num2, num3;
  float average=0;
   cout << "Enter three integer numbers: ";
   // Your code here

  cout << "The average of the numbers is " << average << endl;
  return 0;
}
```

**The output of the your program should be:**

Enter three integer numbers: 22 56 73

The average of the numbers: 50.333333

# FORMATTING FOR PROGRAM OUTPUT

- Besides displaying correct results, a program should present its results attractively with good formats.

- **Stream Manipulators**

  *Stream manipulator* functions are special stream functions that change certain characteristics of the input and output.

  The main advantage of using manipulator functions is they facilitate the formatting of the input and output streams.

# *Stream* manipulator

- ***setiosflags*** **This manipulator is used to control different input and output settings.**

  *setiosflag(ios::fixed)* means the output field will use convertion to a fixed-point decimal notation.

  *setiosflag(ios::showpoint)* means the output field will show the decimal point for floating point number.

  *setiosflag(ios::scientific)* means the output field will use exponential notation.

Note: Without the *ios::fixed* flag, a floating point number is displayed with a default of 6 significiant digits. If the integral part of the number requires more than 6 digits, the display will be in exponential notation.

## Some other format flags for use with *setiosflags()*

| Flag | Meaning |
|---|---|
| *ios::showpos* | display a leading + sign when the number is positive. |
| *ios::dec* | display in decimal format |
| *ios::oct* | display in octal format |
| *ios::left* | left-justify output |
| *ios::right* | right-justify output |
| *ios::hex* | display in hexadecimal format |

To carry out the operations of these manipulators in a user program, you must include the header file <*iomanip.h*>

## Example 3.2.2

```
// a progran to illustrate output conversions
#include <iostream.h>
#include <iomanip.h>
int main()
{
  cout << "The decimal (base 10) value of 15 is " << 15 << endl
       << "The octal (base 8) value of 15 is "
       << setiosflags(ios::oct) << 15 << endl
       << "The hexadecimal (base 16) value of 15 is "
       << setiosflags(ios::hex) << 15 << endl;
    return 0;
  }
```

**The output of the above program**:

```
The decimal (base 10) value of 15 is 15
The octal (base 8) value of 15 is 17
The hexadecimal (base 16) value of 15 is f
```

# Stream Manipulators (cont'd)

- *setw()* The *setw()* stands for *set width*. This manipulator is used to specify the minimum number of the character positions on the output field a variable will consume.

- *setprecision()* The *setprecision()* is used to control the number of digits of an output stream display of a floating point value. *Setprecision(2)* means 2 digits of precision to the right of the decimal point.

**Example**:

```
cout << "|" << setw(10)
     << setioflags(ios::fixed)<< setprecision(3) << 25.67<<"|";
```

**cause the printout**

```
|    25.670|
```

## Example 3.2.1

```
#include <iomanip.h>
int main()
{
  cout << setw(3) << 6 << endl
       << setw(3) << 18 << endl
       << setw(3) << 124 << endl
       << "---\n"
       << (6+18+124) << endl;
  return 0;
}
```

The output of the above program:

```
  6
 18
124
- - -
148
```

# USING MATHEMATICAL LIBRARY FUNCTIONS

- **C++ provides standard library functions that can be included in a program.**

- **If your program uses mathematic function *sqrt()*, it should have the preprocessor command *#include<math.h>* in the beginning of the program.**

| Function Name | Description | Return Value |
|---|---|---|
| abs(a) | Absolute value | Same data type as argument |
| log(a) | Natural logarithm | double |
| sin(a) | sine of a (a in radians) | double |
| cos(a) | cosine of a (a in radians) | double |
| tan(a) | tangent of a (a in radians) | double |

| Function Name | Description | Return Value |
|---|---|---|
| log10(a) | common log (base 10) of a | double |
| pow(a1,a2) | a1 raised to the a2 power | double |
| exp(a) | $e^a$ | double |
| sqrt(a) | square root of a | double |

**Except *abs(a),* they all take an argument of type *double* and return a value of type *double***

**Example 3.3.1**

```
#include <iostream.h>
#include <math.h>
int main()
{
  int height;
  double time;
  height = 800;
  time = sqrt(2 * height / 32.2);    // gravitational constant   g = 32.32
  cout << "It will take " << time << " seconds to fall "
       << height << " feet." << endl;
  return 0;
}
```

**The output of the above program**:

It will take 7.049074 seconds to fall 800 feet.

# Implicit Data Type Conversion

- **Note: *Data type conversion* can take place implicitly across assignment operators, i.e., the value of the expression on the right side is converted to the data type of the variable to the left side.**

- **For example, if *temp* is an integer variable, the assignment *temp* = 25.89 causes the integer value 25 to be stored in the integer variable *temp*.**

# Explicit Data Type Conversion: Casts

- **We have already seen the conversion of an operand's data type within mixed-mode expressions and across assignment operators.**
- **In addition to these implicit data type conversions, C++ also provides for explicit user-specified type conversion. This method is called *casting.***

- ***Casting* or *type casting*, copies the value contained in a variable of one data type into a variable of another data type.**

**The C++ syntax for casting variables is**

*variable = new_type( old_variable);*

**where the *new_type* portion is the keyword representing the type to which you want to cast the variable**.

**Example**:

```
int iNum = 100;
float fNum;
fNum = float(iNum);
```

# Octal and Hexadecimal number

- **To designate an *octal* integer constant, the number must have a leading 0. *Hexadecimal* number are denoted using a leading 0x.**
- **Example**

```
#include <iostream.h>
int main()
{
  cout << "The decimal value of 025 is " << 025 << endl
       << "The decimal value of 0x37 is "<< 0x37 << endl;
  return 0;
}
```

**The output of the above program:**

```
The decimal value of 025 is 21
The decimal value of 0x37 is 55
```

# Block Statement (Compound stat.)

- A block statement = many statements enclosed by parentheses { }
- Any declaration declared within a block only is valid within the block.
- No duplication is allowed in a block
- The extent of a program where a variable can be used is formally referred to as the *scope* of the variable.

- Example:

```
{ // start of outer block
    int a = 25;
    int b = 17;
    cout << “The value of a is “ << a << “ and b is “  << b << endl;
    {   // start of inner block
        float a = 46.25;
        int c = 10;
        cout << “ a is now “ << a << “b is now “ << b
                << “ and c is “ << c << endl;
     }
     cout << “ a is now “ << a
                << “b is now “ << b << endl;
} // end of outer block
```

The output is

```
The value of a is  25   and b is 17
a is now  46.25 b is now 17 and c is 10
a is now  25 b is now  17
```

# Summary

- Two statements: assignment and block
- How to format a value when printing it out
- How to use some library functions
- How to change the type of an expression