## Numeric Data Types

- Integral Types
  - `char`, `short`, `int`, `long`
  - Integer values can be positive or negative, unless they are explicitly declared as `unsigned`, in which case, they can only be positive.

```
char
short
int
long
```





## More Numeric Types

- Floating-Point Types
  - `float`, `double`, `long double`
  - Floating-point types are used to represent real numbers and have both integer and fractional parts. They can also have exponents.

```
float
double
long double
```





## Declarations for Numeric Types

- Named Constant Declarations

```
const float PI = 3.14159;
const float E = 2.71828;
const int MAX_SCORE = 100;
const int MIN_SCORE = -100;
```

- Variable Declarations

```
int studentCount;      // Number of students
int sumOfScores;       // Sum of their scores
float average;         // Average of the scores
```

- Appropriate Assignments

```
average = 95.7;
sumOfScores = 2478;
```





## Arithmetic Operators

+ Unary plus (one operand)
- Unary minus (one operand)
+ Addition (two operands)
- Subtraction (two operands)
* Multiplication (two operands)
/ Floating-point division (floating-point result)
  Integer division (no fractional part)
% Modulus (remainder from integer division) (two operands)

6/2 = 3 7/2 = 3
6 % 2 = 0 7 %2 = 1





## Example Expressions

| Expression | Value |
|---|---|
| 3 + 6 | 9 |





## Example Expressions

| Expression | Value |
|---|---|
| 3 + 6 | 9 |
| 3.4 – 6.1 | -2.7 |

## Example Expressions

| Expression | Value |
|---|---|
| 3 + 6 | 9 |
| 3.4 – 6.1 | -2.7 |
| 2 * 3 | 6 |





## Example Expressions

| Expression | Value |
|---|---|
| 3 + 6 | 9 |
| 3.4 – 6.1 | -2.7 |
| 2 * 3 | 6 |
| 8/2 | 4 |





## Example Expressions

| Expression | Value |
|---|---|
| 3 + 6 | 9 |
| 3.4 – 6.1 | -2.7 |
| 2 * 3 | 6 |
| 8/2 | 4 |
| 8.0/2.0 | 4.0 |





## Example Expressions

| Expression | Value |
|---|---|
| 3 + 6 | 9 |
| 3.4 – 6.1 | -2.7 |
| 2 * 3 | 6 |
| 8/2 | 4 |
| 8.0/2.0 | 4.0 |
| 8/8 | 1 |





## Example Expressions

| Expression | Value |
|---|---|
| 3 + 6 | 9 |
| 3.4 – 6.1 | -2.7 |
| 2 * 3 | 6 |
| 8/2 | 4 |
| 8.0/2.0 | 4.0 |
| 8/8 | 1 |
| 8/9 | 0 |





## Example Expressions

| Expression | Value |
|---|---|
| 3 + 6 | 9 |
| 3.4 – 6.1 | -2.7 |
| 2 * 3 | 6 |
| 8/2 | 4 |
| 8.0/2.0 | 4.0 |
| 8/8 | 1 |
| 8/9 | 0 |
| 8/7 | 1 |

## Example Expressions

| Expression | Value |
|---|---|
| 3 + 6 | 9 |
| 3.4 – 6.1 | -2.7 |
| 2 * 3 | 6 |
| 8/2 | 4 |
| 8.0/2.0 | 4.0 |
| 8/8 | 1 |
| 8/9 | 0 |
| 8/7 | 1 |
| 8 % 8 | 0 |





## Example Expressions

| Expression | Value |
|---|---|
| 3 + 6 | 9 |
| 3.4 – 6.1 | -2.7 |
| 2 * 3 | 6 |
| 8/2 | 4 |
| 8.0/2.0 | 4.0 |
| 8/8 | 1 |
| 8/9 | 0 |
| 8/7 | 1 |
| 8 % 8 | 0 |
| 8 % 9 | 8 |





## Example Expressions

| Expression | Value |
|---|---|
| 3 + 6 | 9 |
| 3.4 – 6.1 | -2.7 |
| 2 * 3 | 6 |
| 8/2 | 4 |
| 8.0/2.0 | 4.0 |
| 8/8 | 1 |
| 8/9 | 0 |
| 8/7 | 1 |
| 8 % 8 | 0 |
| 8 % 9 | 8 |
| 8 % 7 | 1 |





## Example Expressions

| Expression | Value |
|---|---|
| 3 + 6 | 9 |
| 3.4 – 6.1 | -2.7 |
| 2 * 3 | 6 |
| 8/2 | 4 |
| 8.0/2.0 | 4.0 |
| 8/8 | 1 |
| 8/9 | 0 |
| 8/7 | 1 |
| 8 % 8 | 0 |
| 8 % 9 | 8 |
| 8 % 7 | 1 |
| 0 % 7 | 0 |





## Example Expressions

| Expression | Value |
|---|---|
| 3 + 6 | 9 |
| 3.4 – 6.1 | -2.7 |
| 2 * 3 | 6 |
| 8/2 | 4 |
| 8.0/2.0 | 4.0 |
| 8/8 | 1 |
| 8/9 | 0 |
| 8/7 | 1 |
| 8 % 8 | 0 |
| 8 % 9 | 8 |
| 8 % 7 | 1 |
| 0 % 7 | 0 |
| 5 % 2.3 | Error |





## Assignment Statements

- Given `int num; int alpha;`
  `float rate; char ch;`
- Valid Assignments

```
alpha = num + 6;
alpha = num / 2;
num = alpha * 2;
alpha = alpha + 1;
num = num + alpha;
```

## Program Example

```
//**************************************************************
// FreezeBoil program
// This program computes the midpoint between
// the freezing and boiling points of water
//**************************************************************

#include <iostream>

using namespace std;

const float FREEZE_PT = 32.0;   // Freezing point of water
const float BOIL_PT = 212.0;    // Boiling point of water
```





## Program Example (continued)

```
int main()
{
    float avgTemp;            // Holds the result of averaging
                              //   FREEZE_PT and BOIL_PT

    cout << "Water freezes at " << FREEZE_PT << endl;
    cout << " and boils at " << BOIL_PT << " degrees." << endl;

    avgTemp = FREEZE_PT + BOIL_PT;
    avgTemp = avgTemp / 2.0;

    cout << "Halfway between is ";
    cout << avgTemp << " degrees." << endl;

    return 0;
}
```





## Increment and Decrement Operators

++ Increment

-- Decrement

`num++;` is equivalent to `num = num + 1;`

IncrementStatement    Decrement Statement

$\{Variable++;$    $\{Variable--;$





## Compound Arithmetic Expressions

- Precedence Rules
  - Highest: Unary +, Unary –, Parentheses
  - Middle *, /, %
  - Lowest +, -
- Associativity is from left to right

  `int1 – int2 + int3`

  is evaluated

  `(int1 – int2) + int3`





## Precedence Examples

| Expression | Value |
|---|---|
| `10 / 2 * 3` | |





## Precedence Examples

| Expression | Value |
|---|---|
| `10 / 2 * 3 = 5 * 3` | |

UAH CPE 112

## Precedence Examples

| Expression | Value |
|---|---|
| `10 / 2 * 3 = 5 * 3 =` | `15` |



UAH CPE 112

## Precedence Examples

| Expression | Value |
|---|---|
| `10 / 2 * 3 = 5 * 3 =` | `15` |
| `10 % 3 – 4 / 2` | |



UAH CPE 112

## Precedence Examples

| Expression | Value |
|---|---|
| `10 / 2 * 3 = 5 * 3 =` | `15` |
| `10 % 3 – 4 / 2 = 1 – 2` | |



UAH CPE 112

## Precedence Examples

| Expression | Value |
|---|---|
| `10 / 2 * 3 = 5 * 3 =` | `15` |
| `10 % 3 – 4 / 2 = 1 – 2 =` | `-1` |



UAH CPE 112

## Precedence Examples

| Expression | Value |
|---|---|
| `10 / 2 * 3 = 5 * 3 =` | `15` |
| `10 % 3 – 4 / 2 = 1 – 2 =` | `-1` |
| `5.0 * 2.0 / 4.0 * 2.0` | |



UAH CPE 112

## Precedence Examples

| Expression | Value |
|---|---|
| `10 / 2 * 3 = 5 * 3 =` | `15` |
| `10 % 3 – 4 / 2 = 1 – 2 =` | `-1` |
| `5.0 * 2.0 / 4.0 * 2.0`<br>`= 10.0 /4.0 *2.0` | |

## Precedence Examples

| Expression | Value |
|---|---|
| 10 / 2 * 3 = 5 * 3 = | 15 |
| 10 % 3 – 4 / 2 = 1 – 2 = | -1 |
| 5.0 * 2.0 / 4.0 * 2.0 | |
| = 10.0 /4.0 *2.0 | |
| = 2.5 * 2.0 | |





## Precedence Examples

| Expression | Value |
|---|---|
| 10 / 2 * 3 = 5 * 3 = | 15 |
| 10 % 3 – 4 / 2 = 1 – 2 = | -1 |
| 5.0 * 2.0 / 4.0 * 2.0 | |
| = 10.0 /4.0 *2.0 | |
| = 2.5 * 2.0 = | 5.0 |





## Precedence Examples

| Expression | Value |
|---|---|
| 10 / 2 * 3 = 5 * 3 = | 15 |
| 10 % 3 – 4 / 2 = 1 – 2 = | -1 |
| 5.0 * 2.0 / 4.0 * 2.0 | |
| = 10.0 /4.0 *2.0 | |
| = 2.5 * 2.0 = | 5.0 |
| 5.0 * 2.0 / (4.0 * 2.0) | |





## Precedence Examples

| Expression | Value |
|---|---|
| 10 / 2 * 3 = 5 * 3 = | 15 |
| 10 % 3 – 4 / 2 = 1 – 2 = | -1 |
| 5.0 * 2.0 / 4.0 * 2.0 | |
| = 10.0 /4.0 *2.0 | |
| = 2.5 * 2.0 = | 5.0 |
| 5.0 * 2.0 / (4.0 * 2.0) | |
| = 10.0 / 8.0 | |





## Precedence Examples

| Expression | Value |
|---|---|
| 10 / 2 * 3 = 5 * 3 = | 15 |
| 10 % 3 – 4 / 2 = 1 – 2 = | -1 |
| 5.0 * 2.0 / 4.0 * 2.0 | |
| = 10.0 /4.0 *2.0 | |
| = 2.5 * 2.0 = | 5.0 |
| 5.0 * 2.0 / (4.0 * 2.0) | |
| = 10.0 / 8.0 = | 1.25 |





## Precedence Examples

| Expression | Value |
|---|---|
| 10 / 2 * 3 = 5 * 3 = | 15 |
| 10 % 3 – 4 / 2 = 1 – 2 = | -1 |
| 5.0 * 2.0 / 4.0 * 2.0 | |
| = 10.0 /4.0 *2.0 | |
| = 2.5 * 2.0 = | 5.0 |
| 5.0 * 2.0 / (4.0 * 2.0) | |
| = 10.0 / 8.0 = | 1.25 |
| 5.0 + 2.0 / (4.0 * 2.0) | |

UAH CPE 112

## Precedence Examples

| Expression | Value |
|---|---|
| `10 / 2 * 3 = 5 * 3 =` | `15` |
| `10 % 3 – 4 / 2 = 1 – 2 =` | `-1` |
| `5.0 * 2.0 / 4.0 * 2.0` | |
| `= 10.0 /4.0 *2.0` | |
| `= 2.5 * 2.0 =` | `5.0` |
| `5.0 * 2.0 / (4.0 * 2.0)` | |
| `= 10.0 / 8.0 =` | `1.25` |
| `5.0 + 2.0 / (4.0 * 2.0)` | |
| `= 5.0 + 2.0 / 8.0` | |



UAH CPE 112

## Precedence Examples

| Expression | Value |
|---|---|
| `10 / 2 * 3 = 5 * 3 =` | `15` |
| `10 % 3 – 4 / 2 = 1 – 2 =` | `-1` |
| `5.0 * 2.0 / 4.0 * 2.0` | |
| `= 10.0 /4.0 *2.0` | |
| `= 2.5 * 2.0 =` | `5.0` |
| `5.0 * 2.0 / (4.0 * 2.0)` | |
| `= 10.0 / 8.0 =` | `1.25` |
| `5.0 + 2.0 / (4.0 * 2.0)` | |
| `= 5.0 + 2.0 / 8.0` | |
| `= 5.0 + 0.25` | |



UAH CPE 112

## Precedence Examples

| Expression | Value |
|---|---|
| `10 / 2 * 3 = 5 * 3 =` | `15` |
| `10 % 3 – 4 / 2 = 1 – 2 =` | `-1` |
| `5.0 * 2.0 / 4.0 * 2.0` | |
| `= 10.0 /4.0 *2.0` | |
| `= 2.5 * 2.0 =` | `5.0` |
| `5.0 * 2.0 / (4.0 * 2.0)` | |
| `= 10.0 / 8.0 =` | `1.25` |
| `5.0 + 2.0 / (4.0 * 2.0)` | |
| `= 5.0 + 2.0 / 8.0` | |
| `= 5.0 + 0.25 =` | `5.25` |



UAH CPE 112

## Type Coercion (Implicit Conversion)

Integer values and floating-point values are stored differently inside a computer's memory.

Consider the declarations:

```
int someInt;
float someFloat;
```

and the assignments:

```
someFloat = 12;
someInt = 4.8;
```

What actually occurs is

```
someFloat = 12.0
someInt = 4
```



UAH CPE 112

## Type Casting (Explicit Coercion)

- A C++ cast operation consists of a data type name and then, within parentheses, the expression to be converted:

```
someFloat = float (3 * someInt + 2);
someInt = int (5.2 / someFloat –
  anotherFloat);
```

- Countless errors have resulted from unintentional mixing of types.
- It's possible to mix data types within an expression.



UAH CPE 112

## Mixed Type Expression Evaluation

Whenever an integer value and a floating-point value are joined by an operator, implicit type coercion occurs as follows.

1. The integer value is temporarily coerced to a floating-point value.
2. The operation is performed.
3. The result is a floating-point value.

Consider `int sum; int count; float average;`

and `average = sum / count;`

`average` will have the value 0.0

if `sum` = 60 and `count` = 80

UAH CPE 112

## Function Calls

- The following C++ statement has a call to the function `Square` in it:

  `cout << "27 squared is" << Square(27);`
- The function call consists of the symbols

  `Square(27)`

  and may also be called a function invocation.
- The computer temporarily puts the main function on hold and starts the `Square` function running.
- When the `Square` function has finished doing its work, the computer goes back to `main` and picks up where it left off.



UAH CPE 112

## More About Function Calls

- In the above function call, the number 27 is known as an argument (or actual parameter). Arguments make it possible for the same function to work on many different values.
- Syntax Template

  FunctionName ( ArgumentList )
- The argument list is a way for functions to communicate with each other. There can be from zero to many arguments in the list.



UAH CPE 112

## The Last Words on Function Calls For Now

- Value-returning functions
  - The function call is used within an expression; it does not appear as a separate statement.
  - The function computes a value (result) that is then available for use in the expression.
  - The function returns exactly one result – no more, no less.
- The argument to a value-returning function can be any expression of the appropriate type.
- The compiler applies type coercion if the types don't match.



UAH CPE 112

## Library Functions

- Every C++ system includes a large set of prewritten (library) functions.
- To use a function call, place an `#include` directive near the top of your program, specifying the appropriate header file, then make the call.



UAH CPE 112

## Library Function Example

- Example:

```
#include <iostream>
#include <cmath>        // For sqrt() and fabs()

using namespace std;
:
float alpha;
float beta;
:
alpha = sqrt(7.3 + fabs(beta));
```



UAH CPE 112

## Void Functions

- A void function doesn't return a function value, it just performs some action and then quits.
- With a void function, the function call is a separate, stand-alone statement.
- Example call from payroll program

  `CalcPay (payRate, hours, wages);`