Introduction to C++ Programming

Its Applications in Finance

Thanh Hoang

Claremont Graduate University

September 12, 2012

Today Agenda

1. C++ Built-in Data Types
   - Character (*char*)
   - Integer (*int*)
   - Floating Point Types (*float*, *double*)
   - Logical and Relational Type (*bool*)
   - Escape Sequence Codes
2. C++ Variables
   - Fundamental Properties
   - Bits and Bytes
   - Hexadecimal and Octal
3. The *const* Qualifier
4. Data Type Conversion
5. Modulus Operator
6. Summary

C++ Built-in Data Types

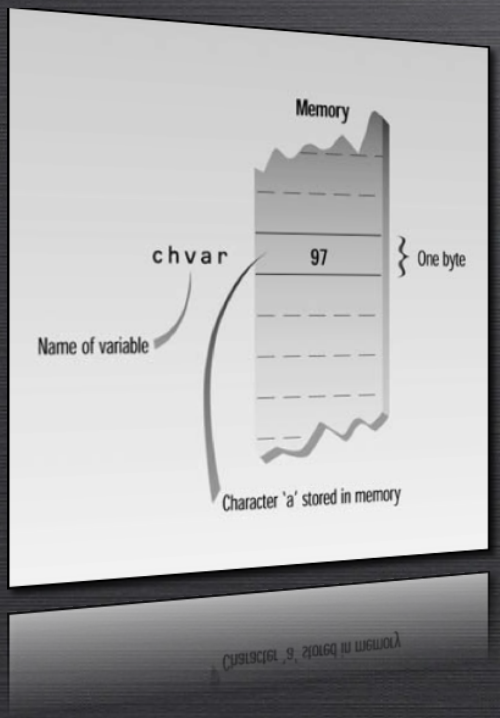
C++ programming language provides built-in data types, including character, integer, floating-point and boolean types.

Type	Meaning
char	Character
int	Integer
float	Floating point
double	Double floating point
bool	Boolean
void	Valueless

Modifiers: *signed*, *unsigned*, *long* and *short*.

Character

Type	Size	Range of Values (*decimal*)
char	1 byte	$-128 \rightarrow +127$ or $0 \rightarrow 255$
unsigned char	1 byte	$0 \rightarrow 255$
signed char	1 byte	$-128 \rightarrow +127$



1. The typical bit width (*32-bit*) of a character is 8, or equivalent to 1 byte.
2. A *char* can hold numbers in the range of 1 byte.
3. A type *char* commonly used to store ASCII characters.
4. Standard C++ also provides a larger character type called *wchar_t*, 16 bits, 0 to 65,535 to handle foreign languages.

Displays the Alphabet

```
// This program displays the alphabet.

#include <iostream>
using namespace std;

int main()
{
     char letter;

     for (letter = "A"; letter <= "Z"; letter++)
          cout << letter;

     cout << endl;

     return 0;
}
```

Constant	Character	Constant Value (ASCII code decimal)
'A'	Capital A	65
'a'	Lowercase a	97 || ' '	Blank	32
'.'	Dot	46
'0'	Digit 0	48
'\0'	Terminating null character	0

Output

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

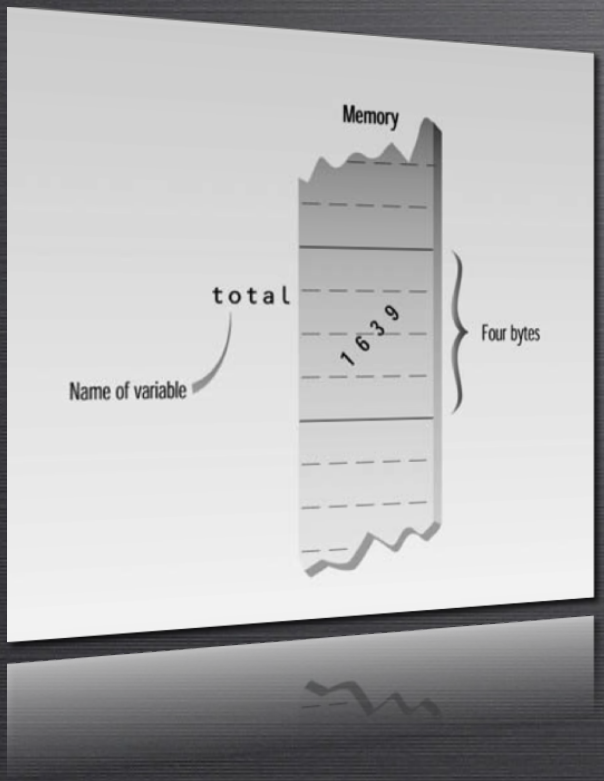
char and *int* Contrasted

```
#include <iostream>
using namespace std;

int main()
{
      // Assign ASCII code for M to ch
      char ch = 'M';
      // Store same code in an int
      int i = ch;

      cout << "The ASCII code for " << ch;
      cout << " is " << i << endl;

      cout << "Add one to the character code:";
      cout << endl;

      ch++; // Change character code in ch
      i = ch; // Save new character code in i

      cout << "The ASCII code for " << ch;
      cout << " is " << i << endl;

      return 0;
}
```

Output

```
The ASCII code for M is 77
Add one to the character code:
The ASCII code for N is 78
```

Integer



Type	Size	Range of Values (decimal)
char	1 byte	$-128 \rightarrow +127$ or $0 \rightarrow 255$
unsigned char	1 byte	$0 \rightarrow 255$
signed char	1 byte	$-128 \rightarrow 127$
int	4 byte	$-2,147,483,648 \rightarrow +2,147,483,647$
unsigned int	4 byte	$0 \rightarrow 4,294,967,295$
short	2 byte	$-32,768 \rightarrow +32,767$
unsigned short	2 byte	$0 \rightarrow 65,535$
long	4 byte	$-2,147,483,648 \rightarrow +2,147,483,647$
unsigned long	4 byte	$0 \rightarrow 4,294,967,295$

Out of Bound (*int*)

```
#include <iostream>
using namespace std;

int main()
{
     short sam = 32767; // -32,768 to 32,767
     unsigned short sue = 32767; // 0 to 65,535

     cout << "Sam has " << sam << " dollars and ";
     cout << "Sue has " << sue << " dollars deposited.\n";
     cout << "Add $1 to each account.\n";
     sam++; sue++; // sam = sam+1, sue = sue+1
     cout << "Now Sam has " << sam << " dollars and ";
     cout << "Sue has " << sue << " dollars deposited.\n";
     cout << "Poor Sam!\n";

     sam = sue = 0;
     cout << "\nSam has " << sam << " dollars and ";
     cout << "Sue has " << sue << " dollars deposited.\n";
     cout << "Take $1 from each account.\n";
     sam--; sue--; // sam = sam-1, sue = sue-1
     cout << "Sam has " << sam << " dollars and ";
     cout << "Sue has " << sue << " dollars deposited.\n";
     cout << "Lucky Sue!\n";

     return 0;
}
```

Output

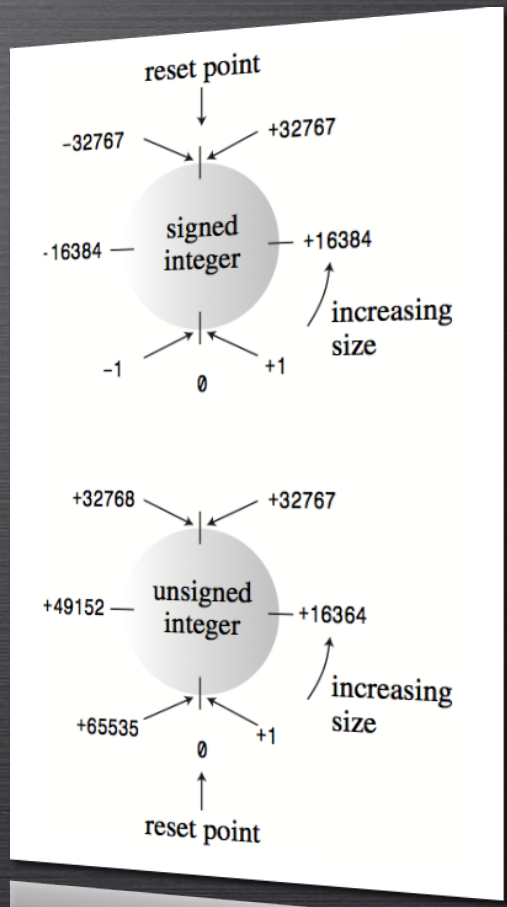
```
Sam has 32767 dollars and
Sue has 32767 dollars deposited.
Add $1 to each account.
Now Sam has -32768 dollars and
Sue has 32768 dollars deposited.
Poor Sam!
Sam has 0 dollars and
Sue has 0 dollars deposited.
Take $1 from each account.
Now Sam has -1 dollars and
Sue has 65535 dollars deposited.
Lucky Sue!
```

Out of Bound (*int*) (*cont.*)

```cpp
/*
    This program shows the difference
    between signed and unsigned integers.
*/

#include <iostream>
using namespace std;

int main()
{
    short i; // -32767 to 32767
    unsigned short j; // 0 to 65535

    j = 45000; // in the range
    i = j; // out of bound

    cout << i << " " << j << endl;

    return 0;
}
```

Output

```
-20536 45000
```



Out of Bound (*int*) (*cont.*)

```cpp
#include <iostream>
using namespace std;

int main()
{
    // −2,147,483,647 to 2,147,483,647
    int var1 = 1500000000;

    // 0 to 4,294,967,295
    unsigned int var2 = 1500000000;

    // Calculation exceeds integer range
    var1 = (var1 * 2) / 3;
    // Calculation within unsigned integer range
    var2 = (var2 * 2) / 3;

    cout << "var1 = " << var1 << endl; // Wrong
    cout << "var2 = " << var2 << endl; // OK

    return 0;
}
```

Output

signedVar = −431655765

unsignedVar = 1000000000

Limits

```cpp
#include <iostream>
#include <climits>
using namespace std;

int main()
{
    // Initialize n_int to max int value
    int n_int = INT_MAX;
    short n_short = SHRT_MAX;
    long n_long = LONG_MAX;

    // sizeof operator yields size of type or of variable
    cout << "int is " << sizeof (int) << " bytes.\n";
    cout << "short int is " << sizeof (n_short) << " bytes.\n";
    cout << "long int is " << sizeof (n_long) << " bytes.\n";
    cout << "Maximum values:\n";
    cout << "int: " << n_int << endl;
    cout << "short int: " << n_short << endl;
    cout << "long int: " << n_long << endl;
    cout << "Minimum int value = " << INT_MIN << endl;
    cout << "Bits per byte = " << CHAR_BIT << endl;

    return 0;
}
```

Output

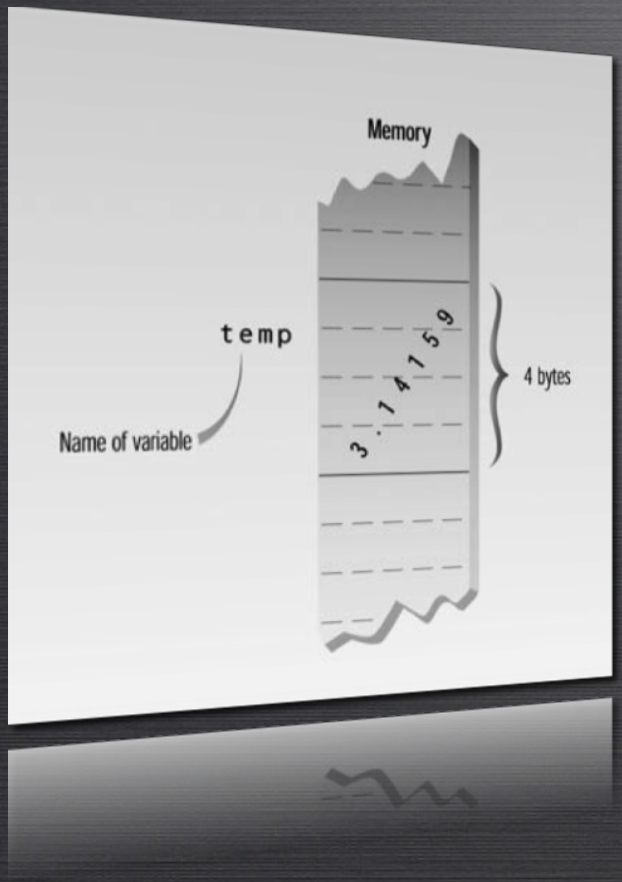
```
int is 4 bytes.
short int is 2 bytes.
long int is 4 bytes.
Maximum values:
int: 2147483647
short int: 32767
long int: 9223372036854775807
Minimum int value = -2147483648
Bits per byte = 8
```

Numeric Data Types Defined by C++

Data Type	Typical Range by ANSI/ISO C++ Standard
char	−128 to 127
unsigned char	0 to 255
signed char	−128 to 127
int	−2,147,483,647 to 2,147,483,647
signed int	−2,147,483,647 to 2,147,483,647
unsigned int	0 to 4,294,967,295
short int	−32,767 to 32,767
signed short int	−32,767 to 32,767
unsigned short int	0 to 65,535
long int	−2,147,483,647 to 2,147,483,647
signed long int	−2,147,483,647 to 2,147,483,647
unsigned long int	0 to 4,294,967,295

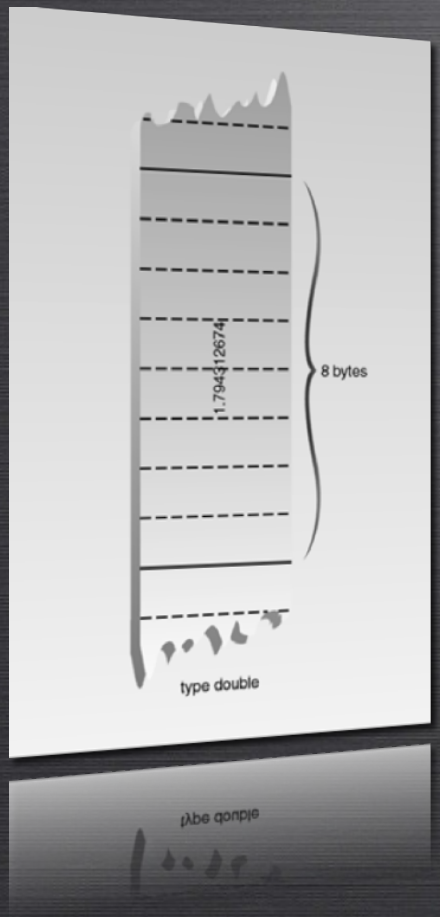
Symbolic Constant	Represents
`SHRT_MAX`	Maximum `short` value
`SHRT_MIN`	Minimum `short` value
`USHRT_MAX`	Maximum `unsigned short` value
`INT_MAX`	Maximum `int` value
`INT_MIN`	Minimum `int` value
`UINT_MAX`	Maximum `unsigned int` value
`LONG_MAX`	Maximum `long` value
`LONG_MIN`	Minimum `long` value
`ULONG_MAX`	Maximum `unsigned long` value

Data Type *float*



1. The typical bit width (*32-bit*) of an *float* is 32, or equivalent to 4 bytes.
2. A *float* variable can hold numbers in the range: -3.4E+38 to 3.4*E*+38, with a precision of 6 digits.
3. For example, *float* radius;
   // variable of type *float*

Data Type *double*



Type	Size	Range of Values	Lowest Positive Value	Accuracy (decimal)
`float`	4 bytes	–3.4E+38	1.2E—38	6 digits
`double`	8 bytes	–1.7E+308	2.3E—308	15 digits
`long double`	10 bytes	–1.1E+4932	3.4E—4932	19 digits

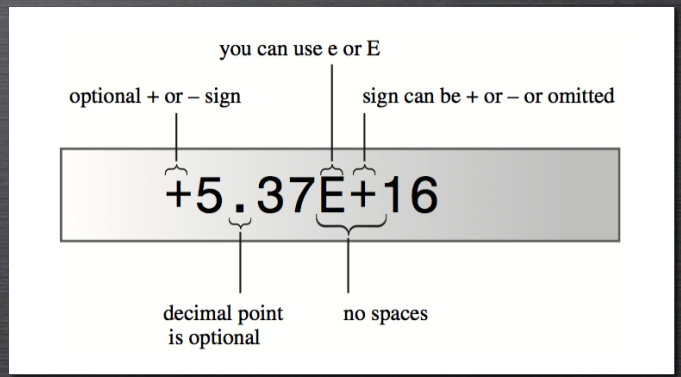
1. The typical bit width (32-bit) of an *double* and *long double* is 64, or equivalent to 8 bytes.
2. A *double* variable can hold numbers in the range: $-1.7E+308$ to $1.7E+308$, with a precision of 15 digits.
3. $1,000,000,000$ can be written as $1.0E+9$ in exponential notation.

Pythagorean Theorem

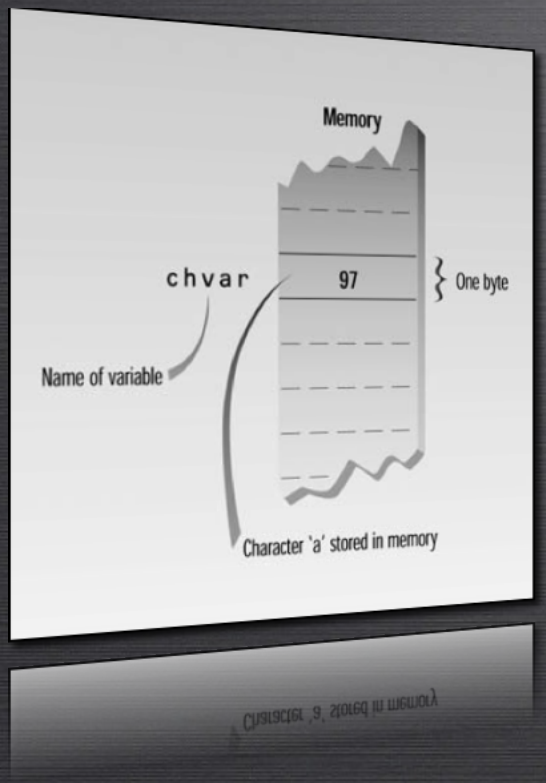
```
/*
     Use the Pythagorean theorem to find
     the length of the hypotenuse given
     the lengths of the two opposing sides.
*/

#include <iostream>
#include <cmath>
using namespace std;

int main()
{
     double x, y, z;
     x = 3.52E+16;
     y = 4.06E+16;

     z = sqrt(pow(x, 2.0) + pow(y, 2.0));

     cout << "Hypotenuse is " << z << endl;

     return 0;
}
```

Output

Hypotenuse is $5.37345e+16$



Data Type *boolean*



1. A *bool* variable can have only two possible values: true and false.
2. A *bool* variable in theory requires only one bit of storage, but in practice compilers often store it as one byte, because a byte can be quickly accessed.
3. For example, is *a* less than *b*? If so, a bool value is given the value true (*non-zero*); otherwise, it is given the value false (*zero*).

An Example of Data Type *bool*

```
// Demonstrate data type bool

#include <iostream>
using namespace std;

int main()
{
     bool b;

     b = false;
     cout << "b is " << b << endl;

     b = true;
     cout << "b is " << b << endl;

     // A bool value can control the if statement
     if (b) cout << "This is executed." << endl;

     // Outcome of a relational operator is a true/false value.
     cout << "10 > 9  is " << (10 > 9) << endl;

     return 0;
}
```

Output

b is 0
b is 1
This is executed.
10 > 9 is 1

Escape Sequence Codes

Single character	Meaning	ASCII code (decimal)
`\a`	alert (BEL)	7
`\b`	backspace (BS)	8
`\t`	horizontal tab (HT)	9
`\n`	line feed (LF)	10
`\v`	vertical tab (VT)	11
`\f`	form feed (FF)	12
`\r`	carriage return (CR)	13
`\"`	" (double quote)	34
`\'`	' (single quote)	39
`\?`	? (question mark)	63
`\\`	\ (backslash)	92
`\0`	string terminating character	0
`\ooo` (up to 3 octal digits)	numerical value of a character	ooo (octal!)
`\xhh` (hexadecimal digits)	numerical value of a character	hh (hexadecimal!)

Escape Sequence Example

```
#include <iostream>
using namespace std;

int main()
{
      long code;

      cout << "\aOperation \"HyperHype\" is now activated!\n";
      cout << "Enter your agent code:________\b\b\b\b\b\b\b\b";
      cin >> code;

      cout << "\aYou entered " << code << "...\n";
      cout << "\aCode verified! Proceed with Plan Z3!\n";

      return 0;
}
```

Output

Operation "HyperHype" is now activated!

Enter your agent code: 12345678

You entered 12345678...

Code verified! Proceed with Plan Z3!

C++ Variables

1. **Three Fundamental Properties:**
   - Where the information is stored
   - What value is kept there
   - What kind of information is stored

2. **C++ Naming Rules:**
   - Alphabetic characters, numeric digits, a underscore character
   - The first letter in a name cannot be a numeric digit
   - Uppercase letters are different from lowercase letters
   - We cannot use a C++ keyword for a name
   - C++ has no limits on the length of a name

Bits and Bytes

1. A *bit* is the fundamental unit of computer memory which we can set to either:
   - One: ON
   - Zero: OFF

2. An 8-bit chunk of memory can be set to 256 different combinations, which represents the values:
   - $0 \rightarrow 255$
   - $-128 \rightarrow +127$

3. A *byte* means an 8-bit unit of memory. A *kilobyte* equals to $1,024$ bytes, and a *megabyte* equals to $1,024$ kilobytes.

Typical Bit Widths and Ranges (*32-bit*)

Data Type	Bit Width	Typical Range in 32-bit Environment
char	8	-127 to 127
unsigned char	8	0 to 255
signed char	8	-127 to 127
int	32	$-2,147,483,647$ to $2,147,483,647$
signed int	32	$-2,147,483,647$ to $2,147,483,647$
unsigned int	32	0 to $4,294,967,295$
short int	16	$-32,767$ to $32,767$
signed short int	16	$-32,767$ to $32,767$
unsigned short int	16	0 to $65,535$
long int	32	$-2,147,483,647$ to $2,147,483,647$
signed long int	32	$-2,147,483,647$ to $2,147,483,647$
unsigned long int	32	0 to $4,294,967,295$
float	32	$3.4E-38$ to $3.4E+38$, with 7 digits of precision
double	64	$1.7E-308$ to $1.7E+308$, with 7 digits of precision
long double	64	$1.7E-308$ to $1.7E+308$, with 7 digits of precision

Hexadecimal and Octal

```
#include <iostream>
using namespace std;

int main()
{
    // Decimal integer constant
    int chest = 42;
    // Hexadecimal integer constant
    int waist = 0x42;
    // Octal integer constant
    int inseam = 042;

    cout << "Monsieur cuts a striking figure!\n";
    cout << "chest = " << chest << endl;
    cout << "waist = " << waist << endl;
    cout << "inseam = " << inseam << endl;

    return 0;
}
```

By default, *cout* displays integers in decimal form, regardless of how they are written in a program. Thus, the following output shows:

Output

```
Monsieur cuts a striking figure!
chest = 42
waist = 66
inseam = 34
```

Display in Hex and Octal

```
#include <iostream>
using namespace std;

int main()
{
      int chest = 42, waist = 42, inseam = 42;

      cout << "Monsieur cuts a striking figure!\n";
      cout << "chest = " << chest;
      cout << " (decimal)\n";
      cout << hex; // Change number base Hex
      cout << "waist = " << waist;
      cout << " (hexadecimal)\n";
      cout << oct; // Changing number base Oct
      cout << "inseam = " << inseam;
      cout << " (octal)\n";

      return 0;
}
```

Output

```
Monsieur cuts a striking figure!
chest = 42 (decimal)
waist = 2a (hexadecimal)
inseam = 52 (octal)
```

The *const* Qualifier

```
#include <iostream>
using namespace std;

#define PI 3.14159

int main()
{
     float rad, area;

     cout << "Enter radius of circle: ";
     cin >> rad;

     // Find the area of a circle
     area = PI * rad * rad;

     cout << "Area is " << area << endl;

     return 0;
}
```

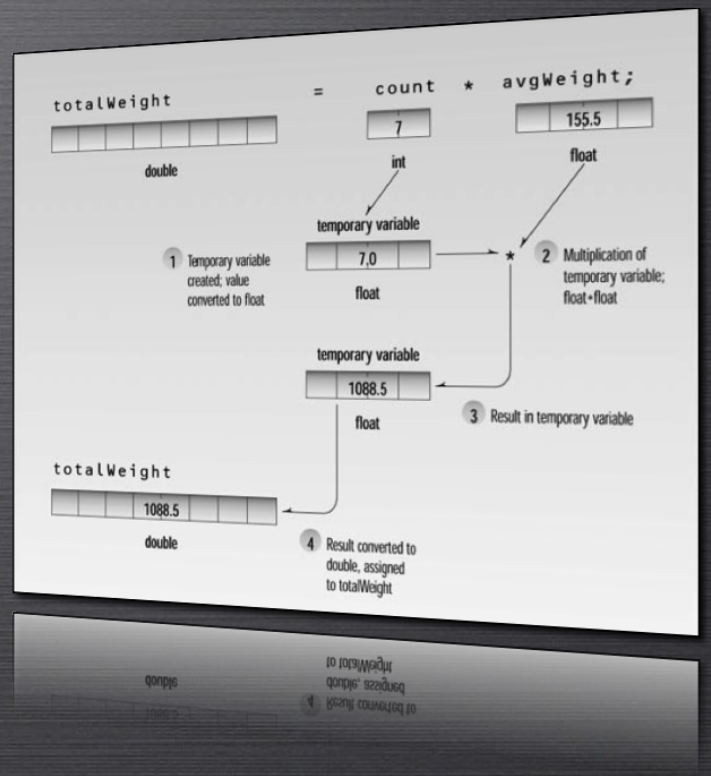
```
#include <iostream>
using namespace std;

const float PI = 3.14159;

int main()
{
     float rad, area;

     cout << "Enter radius of circle: ";
     cin >> rad;

     // Find the area of a circle
     area = PI * rad * rad;

     cout << "Area is " << area << endl;

     return 0;
}
```

Default Type Conversion

```
#include <iostream>
using namespace std;

int main()
{
      int count = 7;
      float avgWeight = 155.5;

      double totalWeight = count * avgWeight;

      cout << "totalWeight = " << totalWeight;
      cout << endl;

      return 0;
}
```

Output

```
totalWeight = 1088.5
```



Cast

General Form

(*type*) expression;

```
#include <iostream>
using namespace std;

int main()
{
     for (int i=1; i <= 10; i++)
     {
          cout << i << "/ 2 is: ";
          cout << (float) i/2 << endl;
     }

     return 0;
}
```

Output

```
1/2 is: 0.5
2/2 is: 1
3/2 is: 1.5
4/2 is: 2
5/2 is: 2.5
6/2 is: 3
7/2 is: 3.5
8/2 is: 4
9/2 is: 4.5
10/2 is: 5
```

Cast Example 1

```
#include <iostream>
using namespace std;

int main()
{
    int intVar = 1500000000;

    intVar = (intVar * 10) / 10; // result too large
    cout << "intVar = " << intVar << endl; // wrong answer

    intVar = 1500000000;

    intVar = ((double) intVar * 10) / 10;
    cout << "intVar = " << intVar << endl; // right answer

    return 0;
}
```

Output

```
intVar = 211509811
intVar = 1500000000
```

Cast Example 2

```
#include <iostream>
using namespace std;

int main()
{
      int a, b;

      // Add two double numbers
      a = 19.99 + 11.99; // without conversion
      b = (int) 19.99 + (int) 11.99;

      cout << "a = " << a << " , b = " << b << endl;

      char ch =  A ;
      cout << "The code for " << ch << " is ";
      cout << int (ch) << endl; // print as integer

      return 0;
}
```

Output

$a = 31$, $b = 30$

The code for A is 65

Modulus Operator

```
// Demonstrate the modulus operator.

#include <iostream>
using namespace std;

int main()
{
      int x, y;

      x = 10;
      y = 3;

      cout << x << " / " << y << " is " << x / y << endl;
      cout << x << " % " << y << " is " << x % y << endl;

      return 0;
}
```

Output

```
10 / 3 is 3
10 % 3 is 1
```

Some Simple Questions

1. **Q:** What is the difference between signed and unsigned integers?

   A: A signed integer can hold both positive and negative values. An unsigned integer can hold only positive values.

2. **Q:** Can a *char* variable be used like a little *integer*?

   A: Yes.

3. **Q:** What values can a *bool* variable have? To what Boolean value does zero convert?

   A: Variables of type bool can be either true or false. Zero = false.

4. **Q:** What does the % operator do? To what types can it be applied?

   A: The % is the modulus operator, which returns the remainder of an integer division. It can be applied to *integer* types.

Some Simple Questions

1. Q: What is the difference between signed and unsigned integers?

   A: A signed integer can hold both positive and negative values. An unsigned integer can hold only positive values.

2. **Q:** Can a *char* variable be used like a little *integer*?

   A: Yes.

3. **Q:** What values can a *bool* variable have? To what Boolean value does zero convert?

   A: Variables of type bool can be either true or false. Zero = false.

4. **Q:** What does the % operator do? To what types can it be applied?

   A: The % is the modulus operator, which returns the remainder of an integer division. It can be applied to *integer* types.

Some Simple Questions

1. Q: What is the difference between signed and unsigned integers?

   A: A signed integer can hold both positive and negative values. An unsigned integer can hold only positive values.

2. Q: Can a *char* variable be used like a little *integer*?

   A: Yes.

3. Q: What values can a *bool* variable have? To what Boolean value does zero convert?

   A: Variables of type bool can be either true or false. Zero = false.

4. Q: What does the % operator do? To what types can it be applied?

   A: The % is the modulus operator, which returns the remainder of an integer division. It can be applied to *integer* types.

Some Simple Questions

1. Q: What is the difference between signed and unsigned integers?

   A: A signed integer can hold both positive and negative values. An unsigned integer can hold only positive values.

2. Q: Can a *char* variable be used like a little *integer*?

   A: Yes.

3. Q: What values can a *bool* variable have? To what Boolean value does zero convert?

   A: Variables of type bool can be either true or false. Zero = false.

4. Q: What does the % operator do? To what types can it be applied?

   A: The % is the modulus operator, which returns the remainder of an integer division. It can be applied to *integer* types.

Some Simple Questions

1. Q: What is the difference between signed and unsigned integers?

   A: A signed integer can hold both positive and negative values. An unsigned integer can hold only positive values.

2. Q: Can a *char* variable be used like a little *integer*?

   A: Yes.

3. Q: What values can a *bool* variable have? To what Boolean value does zero convert?

   A: Variables of type bool can be either true or false. Zero = false.

4. Q: What does the % operator do? To what types can it be applied?

   A: The % is the modulus operator, which returns the remainder of an integer division. It can be applied to *integer* types.

Some Simple Questions

1. Q: What is the difference between signed and unsigned integers?

   A: A signed integer can hold both positive and negative values. An unsigned integer can hold only positive values.

2. Q: Can a *char* variable be used like a little *integer*?

   A: Yes.

3. Q: What values can a *bool* variable have? To what Boolean value does zero convert?

   A: Variables of type bool can be either true or false. Zero = false.

4. Q: What does the % operator do? To what types can it be applied?

   A: The % is the modulus operator, which returns the remainder of an integer division. It can be applied to *integer* types.

Some Simple Questions

1. Q: What is the difference between signed and unsigned integers?

   A: A signed integer can hold both positive and negative values. An unsigned integer can hold only positive values.

2. Q: Can a *char* variable be used like a little *integer*?

   A: Yes.

3. Q: What values can a *bool* variable have? To what Boolean value does zero convert?

   A: Variables of type bool can be either true or false. Zero = false.

4. Q: What does the % operator do? To what types can it be applied?

   A: The % is the modulus operator, which returns the remainder of an integer division. It can be applied to *integer* types.

Some Simple Questions

1. Q: What is the difference between signed and unsigned integers?

   A: A signed integer can hold both positive and negative values. An unsigned integer can hold only positive values.

2. Q: Can a *char* variable be used like a little *integer*?

   A: Yes.

3. Q: What values can a *bool* variable have? To what Boolean value does zero convert?

   A: Variables of type bool can be either true or false. Zero = false.

4. Q: What does the % operator do? To what types can it be applied?

   A: The % is the modulus operator, which returns the remainder of an integer division. It can be applied to *integer* types.

Some Simple Questions

1. Q: What is the difference between signed and unsigned integers?

   A: A signed integer can hold both positive and negative values. An unsigned integer can hold only positive values.

2. Q: Can a *char* variable be used like a little *integer*?

   A: Yes.

3. Q: What values can a *bool* variable have? To what Boolean value does zero convert?

   A: Variables of type bool can be either true or false. Zero = false.

4. Q: What does the % operator do? To what types can it be applied?

   A: The % is the modulus operator, which returns the remainder of an integer division. It can be applied to *integer* types.

Summary

1. C++ Built-in Data Types
   - Character (*char*)
   - Integer (*int*)
   - Floating Point Types (*float*, *double* ...)
   - Logical and Relational Type (*bool*)
   - Escape Sequence Codes
2. C++ Variables
3. The *const* Qualifier
4. Data Type Conversion
5. Modulus Operator
6. Summary

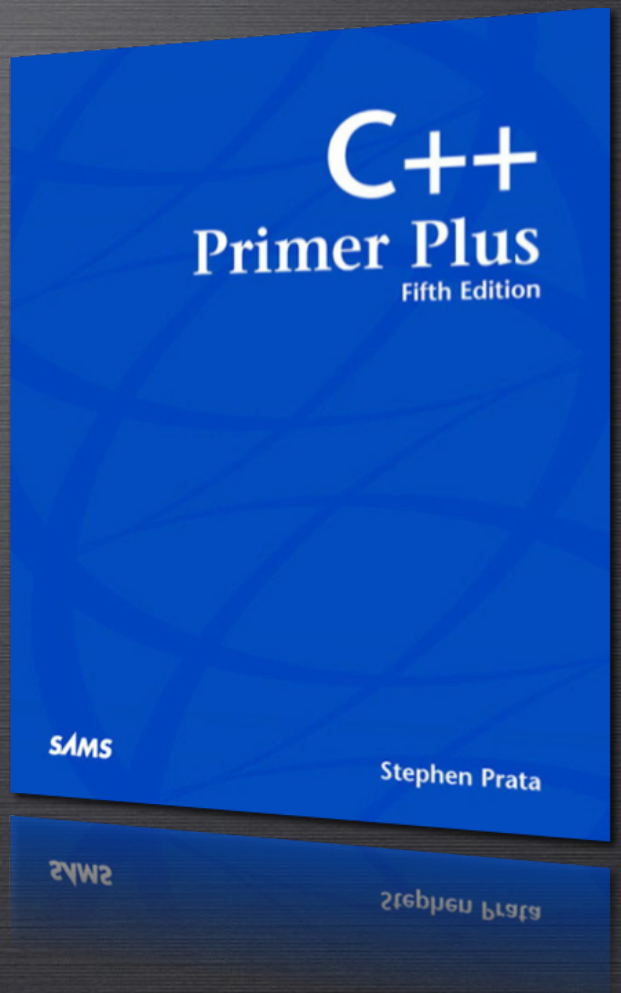
For Further Reading

Stephen Prata.

C++ Primer Plus, 5th Edition,

Chapter 3 and Chapter 5 (Relational Expressions)

SAMS Publishing, 2004.



Weight Conversion

Write a C++ program to use the modulus operator to convert pounds to stone.

1. Given that one stone is 14 pounds.
2. You must use a *const* modifier as a conversion.

Net Present Value

TVBH Corporation is considering an investment of US$50 millions in a capital project that will return after-tax cash flows of US$16 millions per year for the next four years plus another US$20 millions in Year 5. The required rate of return is 10%. By using the Net Present Value (NPV) methodology, we will decide whether this project is profitable or unprofitable.

$$NPV = \sum_{t=1}^{n} \frac{CF_t}{(1+r)^t} - Outlay$$

The decision rule for the NPV is as follow:

- Invest if $NPV > 0$
- Do not invest if $NPV < 0$

Profitability Index

TVBH Corporation investment (discussed earlier) has an outlay of $50 million. By using the Profitability Index (PI), we will decide whether we should invest in this project or not.

$$PI = 1 + \frac{NPV}{Outlay}$$

The investment decision rule for the PI is as follow:

- Invest if $PI > 1.0$
- Do not invest if $PI < 1.0$

Compare NPVs between two projects

TVBH Corporation is considering to invest in two investments: Project A and Project B. Both projects have similar outlays but different pattern of future cash flows. Project A realizes most of its cash payoffs earlier than Project B. For both projects, the required rate of return is 10 percent.

	Year 0	Year 1	Year 2	Year 3	Year 4
Project A	−200	80	80	80	80
Project B	−200	0	0	0	400

If the two projects were not mutually exclusive, what should we do?
Otherwise, if we only have enough capital to invest in one project,
what project should we invest in?

Financial Derivative - Option

Option

In finance, an option is a derivative financial instrument that specifies a contract between two parties for a future transaction on an asset at a reference price (the strike). The buyer of the option gains the right, but not the obligation, to engage in that transaction, while the seller incurs the corresponding obligation to fulfill the transaction.

European Option

European Option is an option that may only be exercised on expiration.

Call Option vs. Put Option

A call option conveys the right to buy something at a specific price; while a put option conveys the right to sell something at a specific price.

Black-Scholes-Merton model

Let's assume the price of the underlying asset, S follows a geometric Brownian Motion process:

$$dS(t) = \mu S(t)dt + \sigma S(t)dW(t) \tag{1}$$

By using Ito's lemma with the assumption of no arbitrage, Black and Scholes showed us the BSM equation to price an European option.

$$c = SN(d_1) - Ke^{-r(T-t)}N(d_2) \tag{2}$$

where:

$$\begin{aligned} d_1 &= \frac{log(\frac{S}{K}) + (r + \frac{1}{2}\sigma^2)\tau}{\sigma\sqrt{\tau}} = \frac{log(\frac{S}{K}) + r\tau}{\sigma\sqrt{\tau}} + \frac{1}{2}\sigma\sqrt{\tau} \\ d_2 &= d_1 - \sigma\sqrt{\tau} \end{aligned}$$

An example of BSM model

The stock price 6 months from the expiration of an option is \$42, the exercise price of the option is \$40, the risk-free interest rate is 10 percent per annum, and the volatility is 20 percent per annum. What prices are European call and put options? *Given that:*

$$\begin{aligned} c &= SN(d_1) - Ke^{-r(T-t)}N(d_2) \\ p &= Ke^{-r(T-t)}N(-d_2) - SN(-d_1) \end{aligned}$$

where:

$$\begin{aligned} d_1 &= \frac{log(\frac{S}{K}) + r\tau}{\sigma\sqrt{\tau}} + \frac{1}{2}\sigma\sqrt{\tau} \\ d_2 &= d_1 - \sigma\sqrt{\tau} \end{aligned}$$