南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

C/C++ Program Design

CS205

Week 2

Prof. Shiqi Yu (于仕琪)

<yusq@sustech.edu.cn>

Prof. Feng Zheng(郑锋)

<zhengf@sustech.edu.cn>

Example for Compile and Link

Compile and Link multiple CPP files

main.cpp

```cpp
#include <iostream>
#include "matrixop.h"

int main()
{
    float pVec1[8] = {1.f, 2.f, 3.f, 4.f, 5.f, 6.f,7.f,8.f};
    float pVec2[8] = {1.f, 2.f, 3.f, 4.f, 5.f, 6.f,7.f,8.f};

    float sum = dotproduct(8, pVec1, pVec2);
    std::cout << "The result is " << sum << std::endl;

    return 0;
}
```

```
g++ -c main.cpp
g++ -c matrixop.cpp
g++ main.o matrixop.o -o dotproduct
./dotproduct
```

matrixop.h

```cpp
#pragma once

float dotproduct(int length, float * pA, float * pB);
```

matrixop.cpp

```cpp
#include "matrixop.h"

float dotproduct(int length, float * pA, float * pB)
{
    float sum = 0;
    for (int i = 0; i < length; i++)
        sum += (pA[i] * pB[i]);

    return sum;
}
```

Integers

The slides are based on the book <Stephen Prata, C++ Primer Plus, 6th Edition, Addison-Wesley Professional, 2011>

Integer Types

- What is the integer?
  - Integers are numbers with no fractional part
- C++ provides several choices
  - `char`
  - `short, int, long, long long`
  - C++ integer types differ in the amount of memory
  - Width is used to describe the differences
  - Both signed and unsigned versions

Width

- Integer Types: width
  - int is 16 bits (the same as short) for older IBM PC implementations
  - int is 32 bits (the same as long) for Windows XP, Windows Vista, Windows 7, Macintosh OS X, VAX, and many other minicomputer implementations
  - The width depends on the platforms (CPU+OS+Complier)

In Example: sizeof Operator

- A important operator: how to use it?
  - A type name
  - A variable name

```
cout << "int is " << sizeof (int) << " bytes.\n";
cout << "short is " << sizeof n_short << " bytes.\n";
```

- What is it used for?
  - To allocate block of memory dynamically
  - To find out number of elements in a array
- May give different output according to machine
- It is a keyword in C Programming

- Run program example

  - What's the max value of your short/int?
  - // limits.cpp -- some integer limits

In Example: Header File-climits

- The climits header file defines symbolic constants
- The compiler manufacturer provides a climits file
- Could you please remember how the preprocessor directives #include and #define work?

```
#define INT_MAX 32767
```

Symbolic Constant	Represents
CHAR_BIT	Number of bits in a char
CHAR_MAX	Maximum char value
CHAR_MIN	Minimum char value
SCHAR_MAX	Maximum signed char value
SCHAR_MIN	Minimum signed char value
UCHAR_MAX	Maximum unsigned char value
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
LLONG_MAX	Maximum long long value
LLONG_MIN	Minimum long long value
ULLONG_MAX	Maximum unsigned long long value

In Example: Initialization

- Initialization combines assignment with declaration
  - Use literal constants: 11101
  - Use macros: INT_MAX
  - Use another variable
- Could you please remember how and why declaration works?
- Initializing the variable when you declare
- Initialization with C++11
  - Using a braced initializer
  - The braces can be left empty
- Run program example
  - // Initialization.cpp-- with C++11

Unsigned Types

- Use unsigned types only for quantities that are never negative
- Increasing the largest value

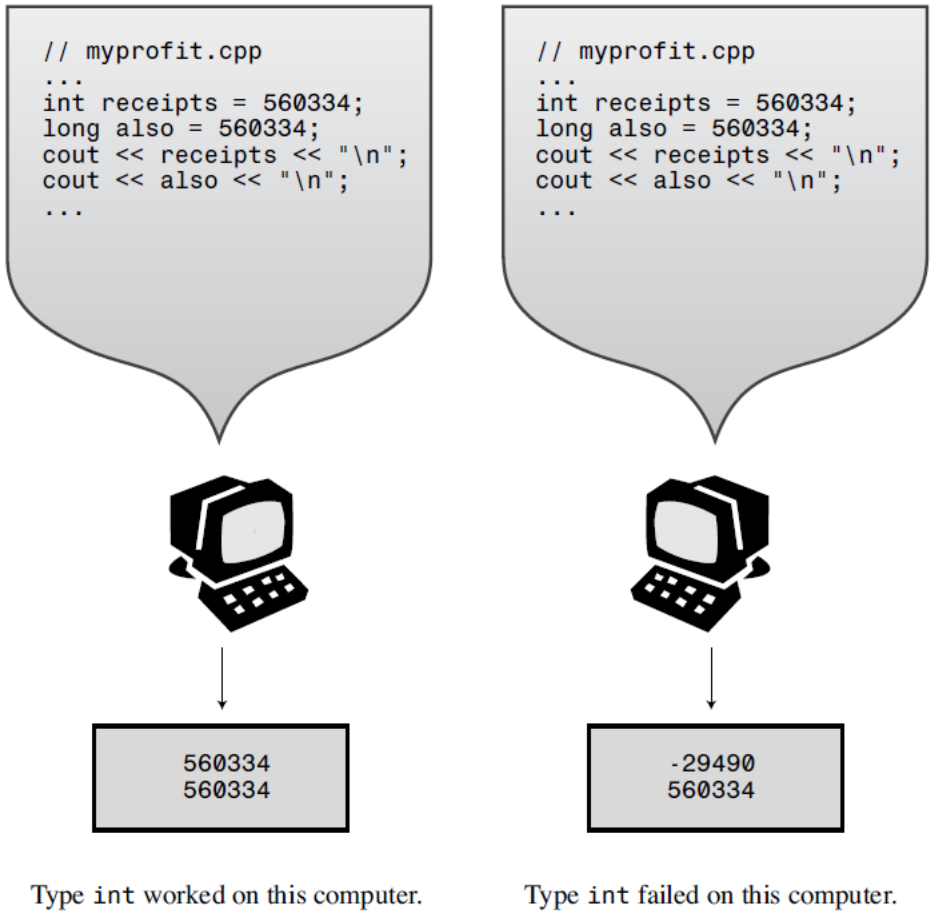
```
unsigned short: 00000000 00000000 (0)
unsigned short: 00000000 00000001 (1)
unsigned short: 01111111 11111111 (32767)
unsigned short: 11111111 11111111 (65535)

signed short: 00000000 00000000 (0)
signed short: 11111111 11111111 (-1)
signed short: 10000000 00000000 (-32768)
```

- Run program example
  - Go beyond the limits for integer types
  - //exceed.cpp -- exceeding some integer limits

Choosing an Integer Type

- The most "natural" integer size: int
- Unsigned int
  - Something that is never negative
  - Integer values need to be too great
- Using short can conserve memory
- If you need only a single byte, you can use char



Type int worked on this computer.

Type int failed on this computer.

Integer Literals

- Number Bases
  - Base 10 (Decimal: the public favorite)
  - Base 8 (Octal: the old Unix favorite)
  - Base 16 (Hexadecimal: the hardware hacker's favorite)
- Uses the first digit or two (Prefix)
  - The first digit is in the range 1–9, the number is base 10
  - The first digit is 0 and the second digit is in the range 1–7, the number is base 8 (octal)
  - The first two characters are 0x or 0X, the number is base 16 (hexadecimal)
- These notations are merely notational conveniences

Examples of Integer Literals

- Run program example
  - //hexoct1.cpp -- shows hex and octal literals
  - cout displays integers in decimal form
- Run program example
  - It provides the dec, hex, and oct manipulators to give cout the messages
  - //hexoct2.cpp -- display values in hex and octal
- For different integral types
  - Suffixes of integer constant

Types allowed for integer literals		
suffix	decimal bases	hexadecimal or octal bases

no suffix	`int`
	`long int`
	`long long int` (since C++11)
		`int`
		`unsigned int`
		`long int`
		`unsigned long int`
		`long long int` (since C++11)
		`unsigned long long int` (since C++11)

u or U	`unsigned int`
	`unsigned long int`
	`unsigned long long int` (since C++11)
		`unsigned int`
		`unsigned long int`
		`unsigned long long int` (since C++11)

l or L	`long int`
	`unsigned long int` (until C++11)
	`long long int` (since C++11)
		`long int`
		`unsigned long int`
		`long long int` (since C++11)
		`unsigned long long int` (since C++11)

both l/L and u/U	`unsigned long int` `unsigned long long int` (since C++11)	`unsigned long int` (since C++11) `unsigned long long int` (since C++11)
ll or LL	`long long int` (since C++11)	`long long int` (since C++11) `unsigned long long int` (since C++11)
both ll/LL and u/U	`unsigned long long int` (since C++11)	`unsigned long long int` (since C++11)

The char Type: Characters and Small Integers

- char type is designed to store characters, such as letters, punctuation and numeric digits
  - Using number codes for letters
  - The char type is another integer type
  - ASCII character set
  - International Unicode character set
- Run program example
  - // chartype.cpp -- the char type
- Run program example
  - // morechar.cpp -- the char type and int type contrasted

	_0	_1	_2	_3	_4	_5	_6	_7	_8	_9	_A	_B	_C	_D	_E	_F
0_ 0	NUL 0000	SOH 0001	STX 0002	ETX 0003	EOT 0004	ENQ 0005	ACK 0006	BEL 0007	BS 0008	HT 0009	LF 000A	VT 000B	FF 000C	CR 000D	SO 000E	SI 000F
1_ 16	DLE 0010	DC1 0011	DC2 0012	DC3 0013	DC4 0014	NAK 0015	SYN 0016	ETB 0017	CAN 0018	EM 0019	SUB 001A	ESC 001B	FS 001C	GS 001D	RS 001E	US 001F
2_ 32	SP 0020	! 0021	" 0022	# 0023	$ 0024	% 0025	& 0026	' 0027	(0028	) 0029	* 002A	+ 002B	, 002C	- 002D	. 002E	/ 002F
3_ 48	0 0030	1 0031	2 0032	3 0033	4 0034	5 0035	6 0036	7 0037	8 0038	9 0039	: 003A	; 003B	< 003C	= 003D	> 003E	? 003F
4_ 64	@ 0040	A 0041	B 0042	C 0043	D 0044	E 0045	F 0046	G 0047	H 0048	I 0049	J 004A	K 004B	L 004C	M 004D	N 004E	O 004F
5_ 80	P 0050	Q 0051	R 0052	S 0053	T 0054	U 0055	V 0056	W 0057	X 0058	Y 0059	Z 005A	[005B	\ 005C	] 005D	^ 005E	_ 005F
6_ 96	` 0060	a 0061	b 0062	c 0063	d 0064	e 0065	f 0066	g 0067	h 0068	i 0069	j 006A	k 006B	l 006C	m 006D	n 006E	o 006F
7_ 112	p 0070	q 0071	r 0072	s 0073	t 0074	u 0075	v 0076	w 0077	x 0078	y 0079	z 007A	{ 007B	| 007C	} 007D	~ 007E	DEL 007F

Letter Number Punctuation Symbol Other Undefined Character changed from 1963 version and/or 1965 draft

char Literals

- Enclose the character in single quotation marks
- There are some characters that you can't enter into a program directly
- Run program example 8
  - // bondini.cpp -- using escape sequences

Character Name	ASCII Symbol	C++ Code	ASCII Decimal Code	ASCII Hex Code
Newline	NL (LF)	\n	10	0xA
Horizontal tab	HT	\t	9	0x9
Vertical tab	VT	\v	11	0xB
Backspace	BS	\b	8	0x8
Carriage return	CR	\r	13	0xD
Alert	BEL	\a	7	0x7
Backslash	\	\\	92	0x5C
Question mark	?	\?	63	0x3F
Single quote	'	\'	39	0x27
Double quote	"	\"	34	0x22

More About char

- signed char ([-128,127]) and unsigned char ([0,255])
  - Allow the compiler developer to best fit the type to the hardware properties
  - Can use signed char or unsigned char explicitly
- wchar_t, char16_t and char32_t
  - wchar_t for wide character type is an integer type

```
wchar_t bob = L'P';          // a wide-character constant
wcout << L"tall" << endl;    // outputting a wide-character string
```

```
char16_t ch1 = u'q';           // basic character in 16-bit form
char32_t ch2 = U'\U0000222B';  // universal character name in 32-bit form
```

The bool Type

- 1 byte NOT 1 bit in storage
- The predefined literals true and false
- The literals true and false can be converted to type int by promotion


```
int ans = true;          // ans assigned 1
int promise = false;     // promise assigned 0
```

- Any nonzero value converts to true, whereas a zero value converts to false

```
bool start = -100;       // start assigned true
bool stop = 0;           // stop assigned false
```

The const Qualifier

- Could you please remember #define directives?

- Use the const keyword to modify a variable declaration and initialization

```
const int Months = 12;  // Months is symbolic constant for 12
```

  - The keyword const is termed a qualifier
  - Note that you initialize a const in the declaration
  - Allow you to specify the type explicitly

```
const int toes;     // value of toes undefined at this point
toes = 10;          // too late!
```

Floating-Point Numbers

The slides are based on the book <Stephen Prata, C++ Primer Plus, 6th Edition, Addison-Wesley Professional, 2011>

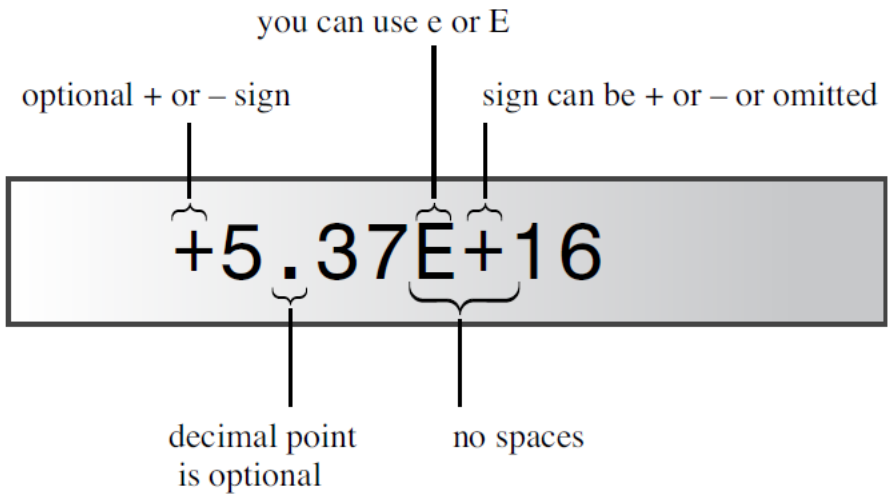
Writing Floating-Point Numbers

- C++ has two ways of writing floating-point numbers
  - Standard decimal-point notation

```
12.34                 // floating-point
939001.32             // floating-point
0.00023               // floating-point
8.0                   // still floating-point
```

  - E notation (*mantissa and exponent*)

```
2.52e+8               // can use E or e, + is optional
8.33E-4               // exponent can be negative
7E5                   // same as 7.0E+05
-18.32e13             // can have + or - sign in front
1.69e12               // 2010 Brazilian public debt in reais
5.98E24               // mass of earth in kilograms
9.11e-31              // mass of an electron in kilograms
```



Precision of Floating-Point Types

- Run program example
  - // floatnum.cpp -- floating-point types
  - setf() forces output to stay in fixed-point notation
  - ios_base::fixed and ios_base::floatfield are constants
  - cout print six figures of digits to the right of decimal point
- Floating-Point Constants
  - By default, floating-point constants is double type

```
1.234f          // a float constant
2.45E20F        // a float constant
2.345324E28     // a double constant
2.2L            // a long double constant
```

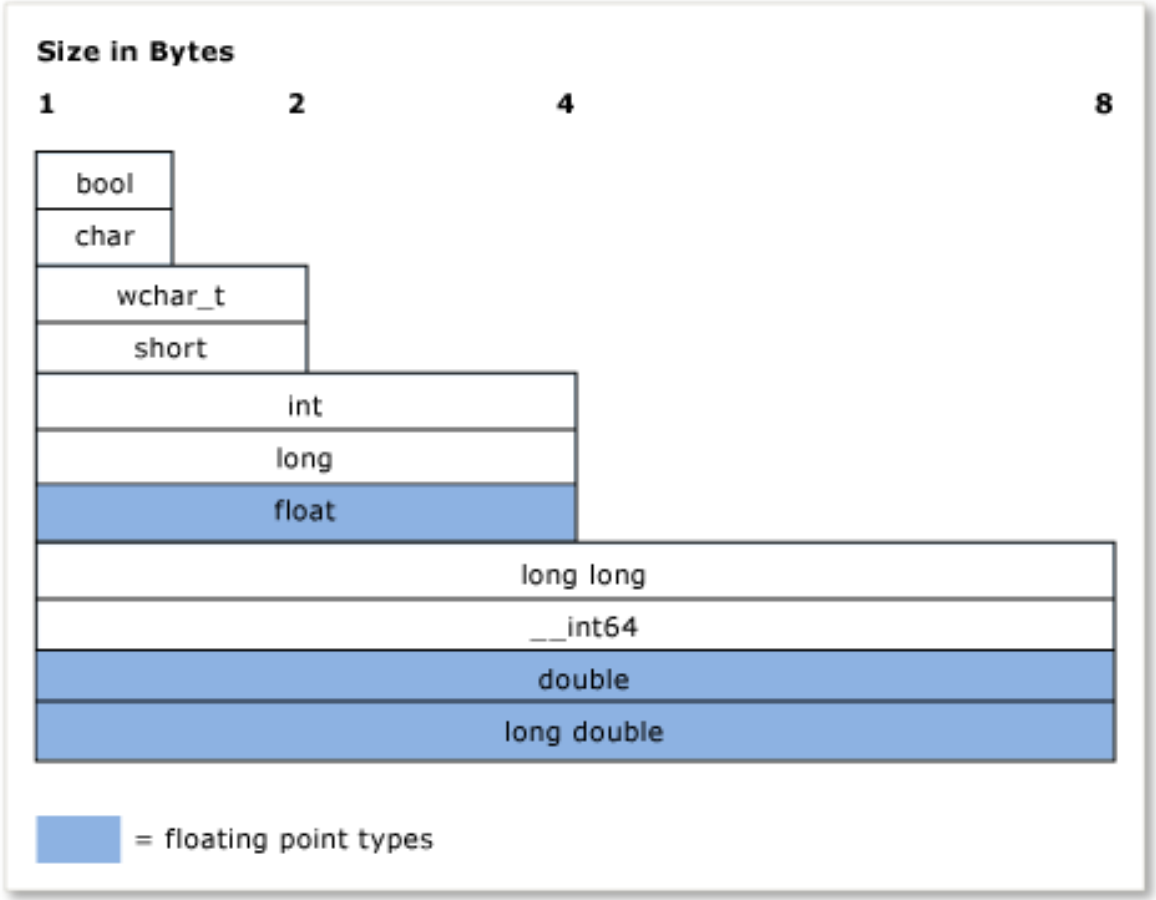
Advantages and Disadvantages of Floating-Point Types

- Advantages
  - Represent values between integers
  - Represent a much greater range of values
- Disadvantages
  - Slightly slower than integer operations
  - Lose precision
- Compare two floating numbers

- Run program example
  - // fltadd.cpp -- precision problems with float
  - Do NOT compare if they are equal as this `(float1==float2)`

Summary of Float and Integer Types

- Arithmetic types
  - Integer types: signed and unsigned
  - Floating-point types



Name	Description	Size*	Range*
char	Character or small integer.	1byte	signed: -128 to 127 unsigned: 0 to 255
short int (short)	Short Integer.	2bytes	signed: -32768 to 32767 unsigned: 0 to 65535
int	Integer.	4bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
long int (long)	Long integer.	4bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
bool	Boolean value. It can take one of two values: true or false.	1byte	true or false
float	Floating point number.	4bytes	+/- 3.4e +/- 38 (~7 digits)
double	Double precision floating point number.	8bytes	+/- 1.7e +/- 308 (~15 digits)
long double	Long double precision floating point number.	8bytes	+/- 1.7e +/- 308 (~15 digits)
wchar_t	Wide character.	2 *or* 4 bytes	1 wide character

C++ Arithmetic Operators

The slides are based on the book <Stephen Prata, C++ Primer Plus, 6th Edition, Addison-Wesley Professional, 2011>

C++ Arithmetic Operators

- C++ uses operators to do arithmetic
- Operators include five basic arithmetic calculations: addition, subtraction, multiplication, division, and taking the modulus: +, -, *, /, %.
- Operators use two values (operands)
- The operator and its operands constitute an expression

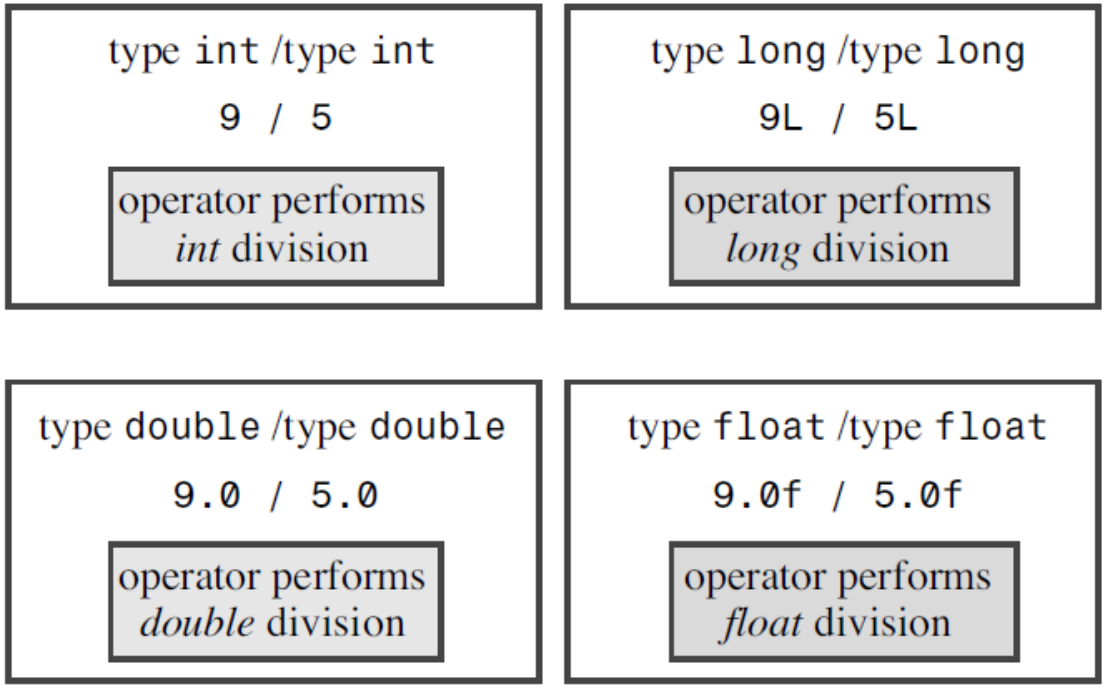
Order of Operation: Operator Precedence and Associativity

- Use precedence rules to decide which operator is used first
  - Usual algebraic precedence
  - Use parentheses to enforce your own priorities
  - Left-to-right associativity or a right-to-left associativity

```
//the result is NOT 17. It's 15. Why???
float logs = 17 / 5 * 5;
```

Four Divisions

- Both operands are integers
  - Perform integer division.
  - Any fractional part of the answer is discarded,
  - Making the result an integer
- One or both operands are floating-point values
  - The fractional part is kept
  - Making the result floating-point
- Four distinct operations
  - int, long, float, and double



The Modulus Operator

- Return the remainder of an integer division
  - Symbol % is used
  - Integer

- Run program example
  - // modulus.cpp -- uses % operator to convert lbs to stone

Type Conversions

- C++ converts values, in cases:
  - Assign a value of one arithmetic type to a variable of another arithmetic type
  - Combine mixed types in expressions
  - Pass arguments to functions
- Result in the loss of some precision
  - double -> float
  - floating-> integer
  - long-> short

Conversion Type	Potential Problems
Bigger floating-point type to smaller floating-point type, such as `double` to `float`	Loss of precision (significant figures); value might be out of range for target type, in which case result is undefined.
Floating-point type to integer type	Loss of fractional part; original value might be out of range for target type, in which case result is undefined.
Bigger integer type to smaller integer type, such as `long` to `short`	Original value might be out of range for target type; typically just the low-order bytes are copied.

Type Conversions in Initialization and Assignment

- C++ uses truncation (discarding the fractional part) and not rounding (finding the closest integer value) when converting floating-point types to integer types
- Run program example
  - // initialization.cpp -- type changes on initialization
- Initialization conversions when {} are used (C++11)

```
const int code = 66;
int x = 66;
char c1 {31325};   // narrowing, not allowed
char c2 = {66};    // allowed because char can hold 66
char c3 {code};    // ditto
char c4 = {x};     // not allowed, x is not constant
x = 31325;
char c5 = x;      // allowed by this form of initialization
```

Automatic Conversions in Expressions

- 1. If either operand is type long double, the other operand is converted to long double.
- 2. Otherwise, if either operand is double, the other operand is converted to double.
- 3. Otherwise, if either operand is float, the other operand is converted to float.
- 4. Otherwise, the operands are integer types and the integral promotions are made.
- 5. In that case, if both operands are signed or if both are unsigned, and one is of lower rank than the other, it is converted to the higher rank.
- 6. Otherwise, one operand is signed and one is unsigned. If the unsigned operand is of higher rank than the signed operand, the latter is converted to the type of the unsigned operand.
- 7. Otherwise, if the signed type can represent all values of the unsigned type, the unsigned operand is converted to the type of the signed type.
- 8. Otherwise, both operands are converted to the unsigned version of the signed type.

Other Conversions

- Conversions in passing arguments for functions
  - C++ promotes char arguments to int

- Type Casts
  - Force type conversions explicitly via the type cast mechanism

```
(long) thorn    // returns a type long conversion of thorn
long (thorn)    // returns a type long conversion of thorn
```

- Run program example
  - // typecast.cpp -- forcing type changes
  - // promotion.cpp -- short is promoted to int

auto Declarations in C++11

- Allow the compiler to deduce a type from the type of an initialization value
  - Compiler assigns the variable the same type as that of the initializer

```
auto n = 100;      // n is int
auto x = 1.5;      // x is double
auto y = 1.3e12L; // y is long double
```

  - STL (Standard **Template** Library)

```
std::vector<double> scores;
std::vector<double>::iterator pv = scores.begin();
std::vector<double> scores;
auto pv = scores.begin();
```

More information https://en.cppreference.com/w/cpp