Chapter 3
Data Organization in Computer Programs

Dr. Le Thanh Sach

Content

- Data and Data Types
- Keywords
- Variable and Variable Declaration
- Scope
- Operators and Expressions
- Enum Data Type
- Constant Definition
- Type Casting
- Exercises

Data and Data Types

- Why do we need data types ?
  - Every program need data for its processing
  - Example:
    - A simple program

```
int main(){
        cout << “LAP TRINH C/C++”;
        return 0;
}
```

      - => Need a place to store data (“LAP TRINH C/C++”) to print to console

Data and Data Types

- Why do we need data types ?
  - Every program need data for its processing
  - Example:
    - A program to solve quadratic equation
      - Data:
        - Coefficient A,B,C of quadratic equation
        - Delta
        - Solution of the equation
    - Human resources management application
      - Data:
        - Identifier, name, salary, etc,....

Data and Data Types

- Why do we need data types ?
  - Every program need data for its processing
  - Programmer needs memory (in RAM) to store data while application is running
    - When users input data (use keyboard, touch on screen, read from sensor,etc..): data will be saved in the memory of RAM
      - Example: Read coefficients A,B,and C of quadratic equation from keyboard
    - During the running: memory is read and processed.
      - Example: To calculate DELTA in quadratic equation, coefficients will be read and used to find DELTA by using this formula (DELTA = B*B – 4*A*C;)

Data and Data Types

- Data Types
  - Describe the data to be processed by programs
  - It specifies:
    - What kind of data can be stored into a variable
    - How many bits of the area should be allocated for a variable
    - Which operators can be applied with the variable
    - …

Data and Data Types

- Data Types
  - Character
  - Boolean: True/False
  - Number
    - Integer
    - Float
  - String: “LAP TRINH C/C++”
  - An array of values
  - Union containing any one of several objects of various types (struct, class)
  - Enumerated data associated with integers (enum)
  - …

Data and Data Types

- How programming languages can distinguish these data types ?
  - C/C++ language (or other languages) defines the meaning of each data type provided by compiler
  - These data types are called fundamental data types
  - These build-in data types are used by using keywords. Programmer should not use these keywords to name other data types or function,....

Data and Data Types

- Data types
  - Fundamental data types
    - Name of type is keyword
    - Meaning of these types defined by programming languages
  - User-defined data type
    - Defined by programmer
    - Meaning of these types defined by programmer based on:
      - Other user-defined data types
      - Fundamental data types
    - Some popular data types in C/C++
      - C: struct, enum
      - C++: class

Data and Data Types

- Data types
  - Fundamental data type
  - User-defined data type
  - Derived data type
    - C/C++ provided symbols to create new data types (based on fundamental or user-defined data types)
    - Example:
      - Array
        - Array of characters, numbers (integer or float), etc...
      - Pointer
        - A pointer to a string, a number or an object,....

Data and Data Types

- Fundamental data types

Type	Keyword	Size	Value
Typeless	void	1 byte	
Character	char	1 byte	‘a’, ‘\n’
Boolean	bool	1 byte	True, false
Real number			
(1)	float	4 bytes	1.5f, 100f
(2)	double	8 bytes	1.5, 100
(3)	long double	12 bytes	1.5l, 100l

Data and Data Types

- Fundamental data types

Type	Keyword	Size	Value
Integer number			
(1)	char, signed char	1 byte	-2, 0, 4
(2)	unsigned char	1 byte	0, 1, 255
(3)	short int, signed short int, short, signed short	2 bytes	10, -100
(4)	unsigned short int, unsigned short	2 bytes	0, 15, 100
(5)	int, signed int, long int, signed long int, long, signed long	4 bytes	10, -100
(6)	unsigned int, unigned long int, unsigned long	4 bytes	0, 15, 100
(7)	long long int, signed long long int	8 bytes	10, -100
(8)	unsigned long long int	8 bytes	0, 15, 100

Data and Data Types

- Fundamental data types

(*) Number of bytes depends on compilers, but will satisfy the following constraints:

1 == sizeof(char) <= sizeof(short) <= sizeof(int)
<= sizeof(long) <= sizeof(long long)

Should use "sizeof" to get the size of a data type!

Data and Data Types

- A program print number of bytes of each data type
  - Function: sizeof(.) return number of bytes of a type

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

int main(){
	//bool
	cout << "sizeof(bool) = " << setw(3) << sizeof(bool) << endl;

	//char
	cout << "char:" << endl;
	cout << "sizeof(char) = " << setw(3) << sizeof(char) << endl;
	cout << "sizeof(signed char) = " << setw(3) << sizeof(signed char) << endl;
	cout << "sizeof(unsigned char) = " << setw(3) << sizeof(unsigned char) << endl;

	//short
	cout << "short:" << endl;
	cout << "sizeof(short) = " << setw(3) << sizeof(short) << endl;
	cout << "sizeof(signed short) = " << setw(3) << sizeof(signed short) << endl;
	cout << "sizeof(unsigned short) = " << setw(3) << sizeof(unsigned short) << endl;

	system("pause");
	return 0;
}
```

Data and Data Types

- Further reading
  - Typedef

Keywords

- What are keywords ?
  - Are words with the special meaning defined by programming language
    - Example: name of fundamental data types
  - Programmer is not allowed to use keywords to name variable, function, constant,etc,...

Keywords

- Keywords in C++

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
continue	for	signed	void
do	if	static	while
default	goto	sizeof	volatile
const	float	short	unsigned

Variable and Variable Declaration

- What is variable ?
  - Is a name of an area in memory, somewhere in virtual memory space.
  - Read / assign value to the area by its name, instead of its address
- How do we use variable ?
  - Variables need to be declared before using (read/write)
  - Compilers allocates memory automatically when meet a variable declaration

Variable and Variable Declaration

- Examples:
  - Create one variable

```
int a;
char c;
```

  - Create many variables in one line

```
int a, b;
char c1, c2;
```

  - Create and assign value

```
int a=10, b;
char c1='A', c2='a';
```

Variable and Variable Declaration

- What do we need when declare a variable ?
  - What type of data will be stored ? → data type of variable
  - What is the meaning of variable ? → name of variable
  - Example:
    - Solve quadratic equation:
      - Coefficients:
        - Data type: float or double. Why?
        - Name: a, b, c. Why?
      - Delta:
        - Data type: float or double. Why?
        - Name: delta
      - Roots of equation:
        - Data type: float or double. Why?
        - Name: x1, x2, s1, s2, sol1, sol2, etc. Why?

Variable and Variable Declaration

- Naming rules (Variable)
  - **Do not use** keywords
  - The first letter:
    - One character (a, A, b, B, …) or underscore (_)
    - **Special characters** are **not** allowed: !,@,#,$,%,^,&,*,(,), …
  - The next characters:
    - digits, letters, underscore

Variable and Variable Declaration

- Example

```
//Chuong trinh giai Phuong trinh bac 2
int main(){
    //Khao bao cac bien
    float a,b,c;
    float delta;
    float x1, x2;
    //Lay a,b,c tu nguoi dung
    //Giai cho truong hop bac 0: a = b = 0
    //Giai cho truong hop bac 1: a = 0
    //Giai cho truong hop 2: a va b <> 0
        //Tinh delta
        //Truong hop: vo nghiem
        //Truong hop: nghiem kep
        //Truong hop: hai nghiem khac nhau
    system("pause");
    return 0;
}
```

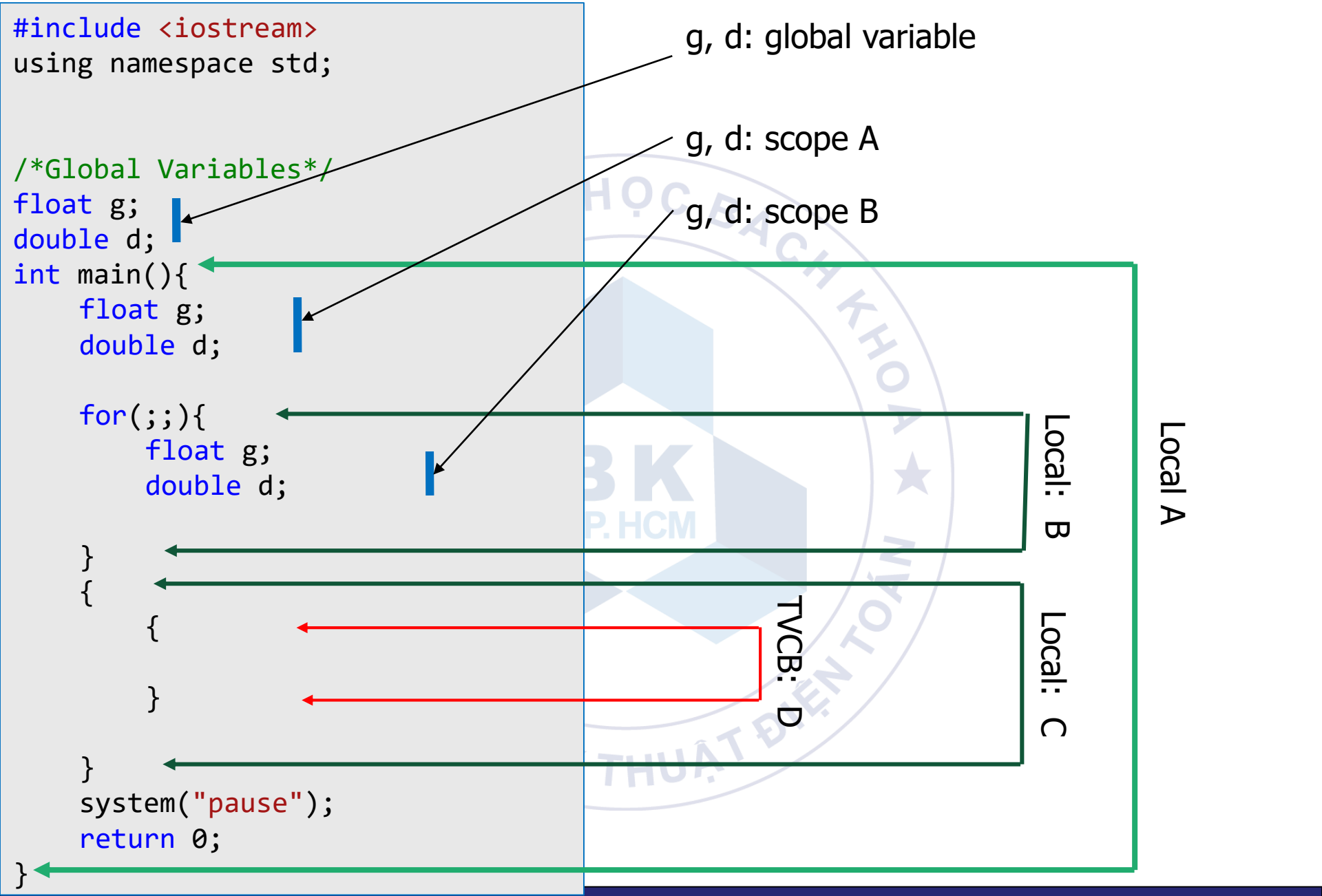
Scope

- What is scope?
  - Is a region of the program where a variable exist and be used.

- There are three places a variable can be declared
  - Global: outside of all function
  - Local:
    - Inside function: from { to } of the body of function
    - Inside a block from { to } of block

  - Function parameters: from { to } of the body

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


/*Global Variables*/
float g;
double d;
int main(){
    float g;
    double d;

    for(;;){
        float g;
        double d;

    }
    {
        {

        }

    }
    system("pause");
    return 0;
}
```



Scope

- Program has 4 scopes naming: A, B, C, D
  - A contains B, C (and D)
  - C contains D
  - Variable g and d (global) can be used in entire program.
    - Be shadowed by inner variable g and d in A
    - Therefore, use the full name to access global variables ::g and ::d
  - Variable g and d in region A can only be used and access in region A
    - However, can not be used in B (why?)
  - Variable g and d in region B
    - Can only be used in B

Scope

- Program has 4 scopes naming: A, B, C and D
  - How can we access global variables (g and d) ?
    - ::g and ::d

Scope

- Initial default value:
  - Local variables: not need initialize default value (depends on compiler)
  - Function parameters: from function call
  - Global variables: based on table below

Data Type	Initial Default Value
int	0
char	'\0'
float	0
double	0
pointer	NULL

Operators and Expressions

- Operators
  - Arithmetic operators
  - Logic operators
  - Bitwise operators
  - Assignment operators
  - Other operators

Operators and Expressions

Operators

Symbol	Meaning	Data type	Example
+	Addition	Numeric	x + y
-	Subtraction	Numeric	x - y
*	Multiplication	Numeric	x * y
/	Division	Numeric	x / y
%	Remainder	Integer or enum type	n % 2
++	Increment	Numeric	++x; y++
--	Decrement	Numeric	--x; y--

Operators and Expressions
Operators

- Increment ++
  - Pre-increment:
    - Example: (++x – y)
      - Increment x by 1
      - Use new value of x in the expression
      - If x = 4 and y = 5
      - (++x – y) will be: 0
  - Post-increment: x++
    - Example: (x++ – y)
      - Use current value of x in the expression
      - Increment x by 1
      - If x = 4 and y = 5
      - (x++ – y) will be: -1
  - In both examples: the final value of x will be 5

Operators and Expressions
Operators

- Decrement --
  - Similar to increment

- When do we use ++ and – ?
  - When we need a variable to control the loop (for, while and do…while)

Operators and Expressions

Logic Operators

Symbol	Meaning	Data type	Example
==	Check equality	Numeric	a == b
!=	Check inequality	Numeric	a != b
>	Larger than	Numeric	a > b
<	Smaller than	Numeric	a < b
>=	Larger or equal	Numeric	a >= b
<=	Smaller or equal	Numeric	a <= b
&&	AND	Logic	b1 && b2
||	OR	Logic	b1 || b2
!	NOT	Logic	!flag

Operators and Expressions

Bitwise Operators

Symbol	Meaning	Data type	Example
&	Bitwise AND	Numeric	a & PATTERN
|	Bitwise inclusive OR	Numeric	a | PATTERN
^	Bitwise exclusive XOR	Numeric	a ^ b
~	One's complement	Numeric	~a
<<	Left shift	Numeric	a << 2
>>	Right shift	Numeric	a >> 2

Operators and Expressions

Assignment Operators

Symbol	Meaning	Data type	Example
=	Assign A = B	All	a = (b + c)*f
+=	A = A+B	Numeric	a += 2
-=	A = A-B	Numeric	a -= 2
*=	A = A*B	Numeric	a *= 2
/=	A = A/B	Numeric	a /= 2
%=	A = A%B	Integer	a %= 2
<<=	A = A << B	Numeric	a <<= 2
>>=	A = A >> B	Numeric	a >>= 2
&=	A = A & B	Numeric	a &= 2
^=	A = A ^ B	Numeric	a ^= 2
|=	A = A | B	Numeric	a |= 2

Operators and Expressions

Other Operators

Symbol	Meaning	Data type
sizeof()	Returns the size (bytes) of a type or a variable	Fundamental data type
&	Returns the address of the memory named Variable	All
*	Returns the value of the memory Pointer points to	All
? :	Conditional operator	(C? A: B) C: Logic expression A, B: expression
[]	Returns the element at the index	Any data type

Operators and Expressions

Priority of Expression

Category	Operator	Associativity
Postfix	() [] -> . ++ - -	Left to right
Unary	+ - ! ~ ++ - - (type) * & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %= >>= <<= &= ^= |=	Right to left
Comma	,	Left to right

Operators and Expressions

- Arithmetic Operators
  - Further reading:
    http://www.tutorialspoint.com/ansi_c/c_operator_types.htm

Operators and Expressions

- Expression
  - Contain operators and operands
    - Expression: X + Y
      - X, Y: Operands
      - + : Operator
      - Addition operator
    - Expression: ! (A && B)
      - (A && B): AND operator
      - ! : not
      - &&: Logic operator
  - The result data type of operation depends on the data types of the operands

Operators and Expressions

- Expression
  - Operands can be constants

Enum Data Type

- Why do we need enum ?
  - In program, a constant (1, 2 or 3,…) should be used to represent:
    - A set of choices
    - A month in a year
    - A color
    - ...
  - To increase code quality (read / understand / maintain), a collection of constants should be grouped together and be used by a name to represent the meaning of this group. In this case, enum type should be used.

Enum Data Type

- Example:

  (1) Color set:

  enum colors {RED, GREEN, BLUE};

  (2) Months set:

  enum months {JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC};

  (3) Choices set:

  enum user_choices {LOAD_DATA, INPUT_DATA, PRINT_DATA};

Enum Data Type

- What is enum ?
  - Is a data type
    - In previous example, we have 3 enum type: colors, months, user_choices.
      - A variable with colors enum type can only be RED, GREEN or BLUE
      - A variable with months enum type can only be JAN, FEB, MAR, etc.
      - A variable with user_choices enum type can only be LOAD_DATA, INPUT_DATA, etc.

Enum Data Type

- What is enum?
  - A collection of constants
    - In previous example, we have collections of constants: colors, months, user_choices.
      - Colors has constants: RED, GREEN, BLUE
      - Months has constants: JAN, FEB, MAR, v.v.
      - User_choices has constants: LOAD_DATA, INPUT_DATA, v.v.
    - The list of values in enum type are assigned a number automatically. The first constant assigned to 0, the second constant assigned to 1, and so on.
    - However, programmer can define the value assigned to constants in the collection, should be the first only.
      - `enum colors {RED = 10, GREEN, BLUE};`

Enum Data Type

- Declare and use enum type

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

int main(){
    enum colors {RED, GREEN, BLUE};
    colors b = BLUE, c = GREEN;
    cout << "b= " << b << endl;
    cout << "c= " << c << endl;

    system("pause");
    return 0;
}
```

Result on console:
b = 2
c = 1
Why ?

Enum Data Type

- How to print name of all constants in enum type ?

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

int main(){
    enum colors {RED, GREEN, BLUE};
    char* colors_names[] = {"RED", "GREEN", "BLUE"};
    colors b = BLUE, c = GREEN;
    cout << "b= " << colors_names[b - RED] << endl;
    cout << "b= " << colors_names[c - RED] << endl;

    system("pause");
    return 0;
}
```

Constant Definition

- What is constant ?
  - Refer to fixed values that the program can not alter during its execution.
  - So,
    - The value of constant must be determined at compile time
- Example
  - (1) Use const keyword

    const int MAX = 50;
  - (2) Use macro

    #define MAX 50
  - (3) Use enum

    enum {MAX};

    enum {MAX = 50};

Constant Definition

- Example
  - Char constant

    const char c = ‘a’;

    const char c = ‘A’;

  - String constant

    const char c[] = “LAP TRINH C/C++”;

    const char c[] = “SAI GON”;

  - Other type constant

    const int a = 100;

    const float f = 10.5f;

    const double d = 10.5;

Type Casting

- What is type casting ?
  - When we assign a value to a variable with the same data type, or operator between two operands with the same data type, there is no type conversion.
    - Example

```
int a = 100;
int b;
b = a;

float f = 10.5f;
float fa;
fa = f;
```

Type Casting

- What is type casting ?
  - But, how can we convert a variable from one data type to another data type ?
  - Example:

```
short a = 100;
Int b;
b = a;

double d = 10.5;
float fa;
fa = d;
```

=> convert the values from one type to another type implicitly (short to int or double to float)

Type Casting

- Type casting operator, in the following forms
  - (type) expression
  - type(expression)
- Two cases:
  - Promotion: smaller to bigger size
  - Truncation: bigger size to smaller size

char < short < int < long < float < double < long double

Type Casting

- Implicit type conversion
  - Is an automatic type conversion by the compiler.
  - Example:

```
short s = 100;
int a = s;  // value will be changed
```

    - Variable a with type “int” is destination data type
    - Variable s with type “short” is source data type
    - “short” similar to “int” so compiler convert type automatically without error

Type Casting

- Implicit type conversion
  - Unchanged source value
    - When source data type has fewer bits than destination data type
    - Example
      - char to short
      - short to int
      - Int to long long
      - Int to double

Type Casting

- Implicit type conversion
  - Unchanged source value
  - Changed source value
    - unsigned to signed | signed to unsigned
      - Use two's complement rule
        - Two's complement = one's complement + 1
        - One's complement = (0 → 1, 1 → 0)
    - Logic to number:
      - true → 1
      - false → 0
    - Number to logic
      - Khác 0 → true
      - Bằng 0 → false

Type Casting

- Implicit type conversion
  - Unchanged source value
  - Changed source value
    - unsigned to signed | signed to unsigned
    - Logic to number
    - Number to logic
    - float or double to integer (char, short, int, v.v)
      - Fractional part will be cut (2.456 → 2)
      - If integral part is out of range of destination data type then nothing will be known ! (based on compiler)

Type Casting

- Explicit type conversion
  - Is special programming instruction which specifies what data type (decided by programmer) to treat a variable in a given expression.
  - How to use ?
    - Functional notation

      double x = 10.5;

      int a = int(x);
    - C-like cast notation

      double x = 10.5;

      int a = (int) x;
  - Example
    - Find values of x and y in the following example

Type Casting

- Type casting

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

int main(){
    double x = 3/2;
    double y = (double)3/2;

    cout << "x = ", << setw(4) << setprecision(2) << x << endl;
    cout << "y = ", << setw(4) << setprecision(2) << y << endl;

    system("pause");
    return 0;
}
```

Type Casting

- Type casting
  - Example
    - Find values of a and b and result on the console in the following example

Type Casting

- Type casting

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

int main(){
    unsigned char a;
    signed char b;
    a = -1;
    b = 255;

    cout << "a= " << static_cast<unsigned>(a) << endl;
    cout << "b= " << static_cast<unsigned>(b) << endl;
    cout << "a < 5 is "; a < 5 ? cout << "true \n": cout << "false \n";
    cout << "b > 5 is "; b > 5 ? cout << "true \n": cout << "false \n";

    system("pause");
    return 0;
}
```

Exercises

Summary

- How to handle data in a C++ program
  - Data types
    - Built-in: char, int, float, double, …, void
    - Derived: array, pointer, structure, union, enum
  - → enum and structure for abstract data
  - → More advanced types (array, pointer) come later.
  - Variables: declaration vs. definition
    - Naming
    - Type
    - Value
    - Scope

Summary

- Constants
  - No change during the execution of the program
  - Known at the compile time
  - Defined with: #define, enum, const
- Expressions
- Operators
  - Assignment
  - Arithmetic
  - Bitwise
  - Logic
  - Relational and others
- Type casting: explicit vs. implicit conversion