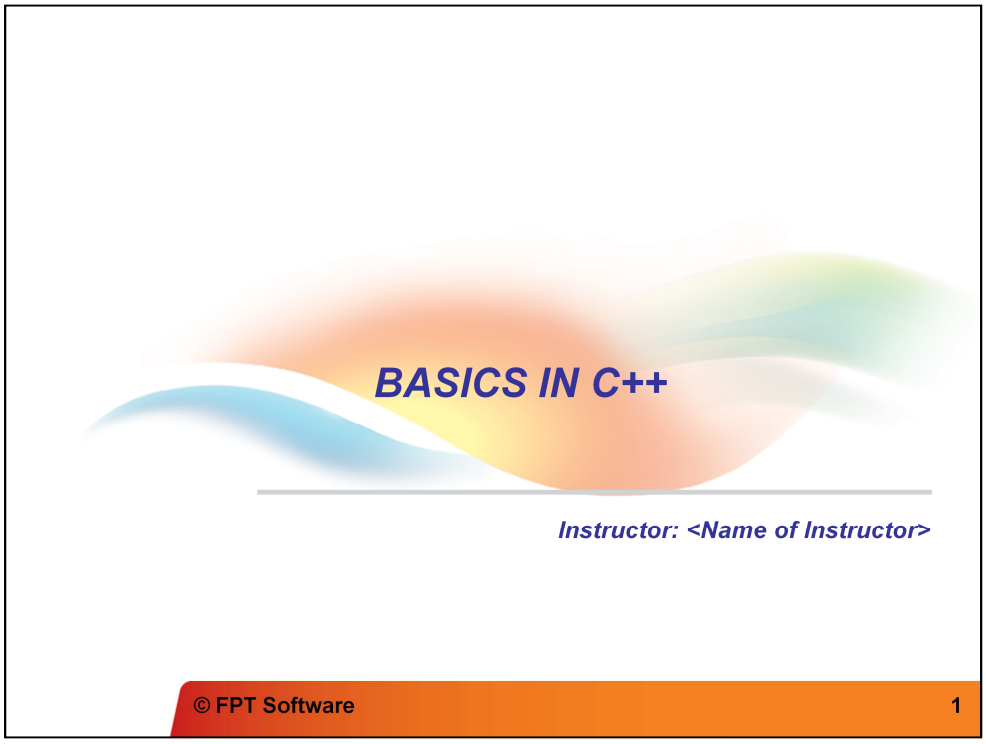


Latest updated by: HanhTT1

Agenda

- Introduction to C++
- Variables & Data types
- Namespaces
- Operators and Operands
- Introduction to function
- Exploring functions
- Logical comparisons
- Conditional statements
- Constructing Expressions
- Shallow copy vs Deep copy

Introduction to C++

- Header files: C++ is a huge language, to make writing code easy, some of functions is written already called libraries, these libraries are placed at the beginning of the program, they are header files with extension ".h". To use a library in a program must use "include" before the name of library, example:

  ```
  #include "book.h"
  ```

- Namespaces: A namespace is a section of code, because C++ is so huge, its libraries are created in different namespaces. To use a library in a namespace must specify library and its namespace, example:

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

Introduction to C++ (2)

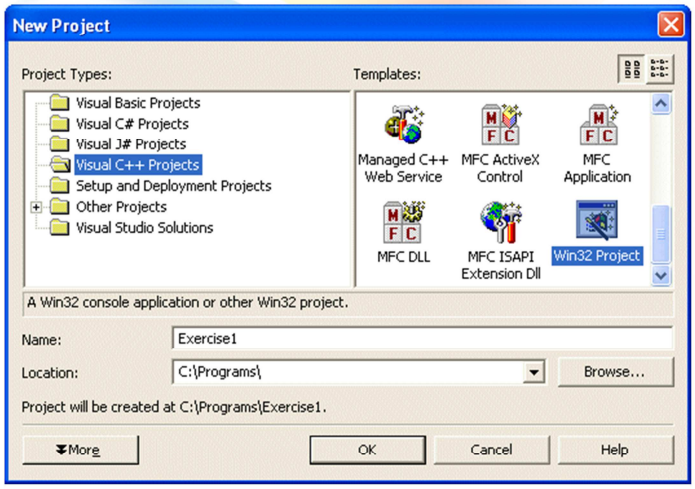
- C++ Projects: basically, a C++ project likes project in C, it includes primary function called main() and set of functions and libraries. Example:

```
#include <iostream>
using namespace std;
main(){}
```

First project with VC++ NET

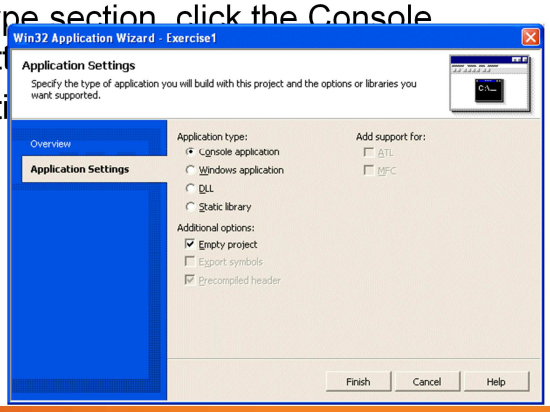
- Start Microsoft Visual Studio .NET.
- On the main menu of Microsoft Development Environment, click File -> New -> Project...
- On the New Project dialog box, in the Location box (bottom of the dialog box), type a drive followed by a folder name such as C:\Programs\MSVC .Net
- In the Project Type, click Visual C++ Projects
- In the Templates list view, click Win32 Project
- In the Name box, type Exercise1

First project with VC++ NET (2)



First project with VC++ NET (3)

- Click OK.
- In the Win32 Application Wizard - Exercise1 dialog box, click Application Settings
- In the Application Type section, click the Console Application radio but
- In the Additional Opti Project check box



First project with VC++ NET (4)

- Click Finish.
- To create a new C++ file, on the main menu, click Project -> Add New Item... Or, on the Solution Explorer, right-click Exercise1 -> Add and click Add New Item...
- In the Categories list, make sure Visual C++ or C++ is selected.
- In the Templates list view, click C++ File (.cpp)
- In the Name box, replace the contents with Exercise

First project with VC++ NET (5)

- Click Open From what we know about C++ already, change the contents of the file with:

```
#include <iostream>
using namespace std;
int main()
{
	cout << "C++ is fun!!!\n";
	return 0;
}
```

First project with VC++ NET (6)

- To execute the program, on the main menu, click Build -> Execute Exercise1.exe
- When asked to save the project, click Yes
- After viewing the result, press Enter to close the DOS window and return to MSVC

Keywords

C++ Reserved Words

asm	auto	bad_cast	bad_typeid	bool
break	case	catch	char	class
const	const_cast	continue	default	delete
do	double	dynamic_cast	else	enum
except	explicit	extern	false	finally
float	for	friend	goto	if
inline	int	long	mutable	namespace
new	operator	private	protected	public
register	reinterpret_cast	return	short	signed
sizeof	static	static_cast	unsigned	struct
switch	template	this	throw	true
try	type_info	typedef	typeid	typename
union	unsigned	using	virtual	void
volatile	wchar_t	while		

Variables

- In C++, variables can be declared at any place in program before using
- The name of a variable:
  - ✓ Starts with an underscore "_" or a letter, lowercase or uppercase, such as a letter from a to z or from A to Z. Examples are Name, gender, _Students, pRice
  - ✓ Can include letters, underscore, or digits. Examples are: keyboard, Master, Junction, Player1, total_grade, _Score_Side1
  - ✓ Cannot include special characters such as !, %,], or $
  - ✓ Cannot include an empty space
  - ✓ Cannot be any of the reserved words
  - ✓ Should not be longer than 32 characters (although allowed)
- C++ is case sensitive

Variable scope

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

double num; //Global variable

int main() {
     {
             double number = 3.14159;      //local variable
             cout << "Number = " << number << "\n";
     }

     {
             double number = 2.98;  //local variable
             cout << "Number = " << number << "\n";
     }
     return 0;
}
```

Static variables

```
#include <iostream>
using namespace std;
void Starter(int y)
{
    static double a = 100;
    static double b = 200;
    a = a / y;
    b = b + 2;
    cout << "y = " << y << endl;
    cout << "a = " << a << endl;
    cout << "b = " << b << endl;
    cout << "b / a = " << b / a << "\n\n";
}

int main()
{
    Starter(2);
    Starter(5);
    return 0;
}
```

Operators and Operands

- Math operators: +, -, *, /, %, ++, --, +=, -=, *=, /=
- Comparison operators: <, >, !=, ==, >=, <=
- Logical operators: &&, ||, !
- Conditional operators:
  <condition> ? <true expression> : < false expression>

Data Types

Type	Low	High	Bytes
char	-128	127	1
short	-32768	32767	2
int	-2147483648	2147483647	4
long	-2147483648	2147483647	4
float	3.4×10^{-38}	3.4×10^{38}	4
double	1.7×10^{-308}	1.7×10^{308}	8
long double	3.4×10^{-4932}	3.4×10^{4932}	10

String type

- String is not a basic data type in C++, it is defined in string standard class

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

int main () {
   string mystring;
   mystring = "This is the initial string content";
   cout << mystring << endl;
   mystring = "This is a different string content";
   cout << mystring << endl;
   return 0;
}
```

Casting

- C casting:

```
main() {
    cout << "Number: " << (int) 3.14159 << "\n";
    return 0;
 }
```

- C++ casting: C++ provides its own support of value casting using variable keywords so you can specify the type of conversion you want. One of the keywords used is **static_cast** and the formula is:

  static_cast<*DataType*>(*Expression*)

```
#include <iostream>
using namespace std;
int main() {
     cout << "Number: " << static_cast<int> (3.14159) << "\n";
     return 0;
}
```

Namespaces (1)

- Creating namespace:

```
namespace Mine {
    int a;
}
```

- Accessing a namespace:

```
#include <iostream>
using namespace std;
namespace Mine {
    int a;
}
int main() {
   Mine::a = 140;
   cout << "Value of a = " << Mine::a << endl;
   return 0;
}
```

Namespaces (2)

- Using keyword to access a namespace:

```
#include <iostream>
using namespace std;
namespace InterestAndDiscount {
      double principal;
      double rate;
      int periods;
}
int main() {
      using namespace InterestAndDiscount;
      double interest;
      double maturityValue;
      cout << "Interest and Discount\n";
      cout << "Principal: $";
      cin >> principal;
      cout << "Rate (example 8.75): ";
      return 0;
}
```

Function

- Calling function:

```
#include <iostream>
using namespace std;
void Message() {
     cout << "This is C++ in its truest form.";
}
int main() {
     Message(); // Calling the Message() function
     return 0;
}
```

Function and Arguments

- Passing arguments by values

```
double GetHours(string FullName) {
      cout << "Full name is: " << FullName;
}
int main() {
      string FullName = "Michel mata";
      double Hours;
      Hours = GetHours(FullName);
      return 0;
}
```

- Passing arguments by reference

```
 void swap (int & a, int& b)  {
     int tmp;
     tmp = a;
     a = b;
     b = tmp;
 }
 int main() {
    int x=3;
    int y=5;
    swap(x,y);
    return 0;
 }
```

Function Overloading

```
#include <iostream>
using namespace std;
// Rectangle
double MomentOfInertia(double b, double h) {
    return b * h * h * h / 3;
}
// Semi-Circle
double MomentOfInertia(double R) {
    const double PI = 3.14159;
    return R * R * R * R * PI/ 8;
}
int main() {
    double base, height, radius;
    cout << "Enter the dimensions of the Rectangle\n";
    cout << "Base: ";
    cin >> base;
    cout << "Height: ";
    cin >> height;
    cout << "\nMoment of inertia with regard to the X axis: ";
    cout << "I = " << MomentOfInertia(base, height) << "mm";
    cout << "\n\nEnter the radius: ";
    cin >> radius; cout << "Moment of inertia of a semi-circle with regard to the X axis: ";
    cout << "I = " << MomentOfInertia(radius) << "mm\n\n";
    return 0;
}
```

Inline Function

```
#include <iostream>
using namespace std;
inline void Area(float Side) {
  cout << "The area of the square is " << Side * Side;
}
int main() {
  float s;
  cout << "Enter the side of the square: ";
  cin >> s;
  Area(s);
  return 0;
}
```

Static function

```
#include <iostream>
using namespace std;
static int GetNumberOfPages() {
  int pages = 842;
  return pages;
}
int main( int argc, char * argv[]) {
  cout << "This book contains " <<
  GetNumberOfPages() << " pages";
  return 0;
}
```

Conditional statements

- If … else …. Statement

```
if(Condition)
   Statement1;
else
   Statement2;
```

- Switch statement:

```
switch(Expression) {
   case Choice1:
      Statement1;
   case Choice2:
      Statement2;
   case Choice-n:
      Statement-n;
}
```

- Looping decisions:

```
while(Condition) Statement;
do Statement while (Condition);
for( Start; End; Frequency) Statement;
```

Structure

```
struct employee {
        char name [ 30 ];
        int age;
        double pension;
        float salary;
        char sex;
        struct date hire_date;
    } reg_employees[100 ];

int main() {
   int i;
   for ( i = 0; i < 100; i++) {
        cout << "Age is " << reg_employees[i].age;
   }
   return 0;
}
```

Array

- Declare an array:

  Syntax: *DataType ArrayName[dimension]*

  Example:

```
int age[12];
float grade[100];
double distance[] = {44.14, 720.52, 96.08, 468.78, 6.28};
```

- Array and Function:

```
void DisplayTheArray(double member[5]) {
      for(int i = 0; i < 5; ++i)
               cout << "\nDistance " << i + 1 << ": " << member[i];
      cout << endl;
}
void DisplayTheArray(double member[]) {
      for(int i = 0; i < 5; ++i)
               cout << "\nDistance " << i + 1 << ": " << member[i];
      cout << endl;
}
```

Pointer Introduction

- Pointer variable store address of memory area

```
int *pointer;
```

- Pointer to pointer:

```
#include <iostream>
using namespace std;
int main() {
    int value = 26;
    int *pointer;
    int **pointerToPointer;
    pointer = &value;
    pointerToPointer = &pointer;
}
```

Passing pointer to function

- Passing pointer to function:

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

void Pickup(int *sht, int *pt)     {
     *sht = 26;
     *pt = 17;
     cout << "\nWithin Pickup()" << "\n\tShirts = " << *sht <<
"\n\tPants  = " << *pt;
}
int main() {
     int shirts = 12;
     int pants = 5;
     Pickup(&shirts, &pants);
     return 0;
}
```

Return a pointer from a function

```
include <iostream>
using namespace std;
int main() {
   int *GetNumber();
   int *number;
   number = GetNumber();
   cout << "Number = " << *number << endl;
   return 0;
}
int *GetNumber() {
   int *a = new int(2885);
   return a;
}
```

Array and Pointer

- Relating a Pointer to an Array

```
#include <iostream>
using namespace std;
int main() {
      int number[] = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
      int *pNumbers = Number;
      cout << "Addresses";
      cout << "\n Number : " << Number;
      cout << "\npNumbers : " << pNumbers;
       cout << "\npNumbers + 1: " << pNumbers + 1;
       cout << "\n*(pNumbers+2): " << *(pNumbers+2);
      return 0;
}
```

Result:
Addresses
Number : 1245020
pNumbers : 1245020
pNumber +1 : 1245024
*(pNumbers+2): 31

A Pointer as parameter

```
#include <iostream>
using namespace std;
double CalculateNetPrice(double *disc);
int main()
{
   double finalPrice;
   double discount = 20;
   finalPrice = CalculateNetPrice(&discount);
   cout << "\nAfter applying a 20% discount";
   cout << "\nFinal Price = " << finalPrice << "\n";
   return 0;
}
double CalculateNetPrice(double *discount)
{
   double origPrice;
   cout << "Please enter the original price: ";
   cin >> origPrice;
   return origPrice - (origPrice * *discount / 100);
}
```

Result ????

Pointers and Multi-Dimensional Arrays

```
int main()
{
    int number[2][6] = { { 31, 28, 31, 30, 31, 30 }, { 31, 31, 30, 31, 30, 31 } };
    int *pNumbers[2];
    *pNumbers = number[0];
    (*pNumbers)[0] = number[0][0];
    (*pNumbers)[1] = number[0][1];
    (*pNumbers)[2] = number[0][2];
    *(pNumbers+1) = number[1];
    (*(pNumbers+1))[0] = number[1][0];
    (*(pNumbers+1))[1] = number[1][1];
    (*(pNumbers+1))[2] = number[1][2];
    cout << "List of Numbers";
    cout << "\n(*pNumbers)[0] = " << (*pNumbers)[0];
    cout << "\n(*pNumbers)[1] = " << (*pNumbers)[1];
    cout << "\n(*pNumbers)[2] = " << (*pNumbers)[2];
    cout << "\n(*(pNumbers+1))[0] = " << (*(pNumbers+1))[0];
    cout << "\n(*(pNumbers+1))[1] = " << (*(pNumbers+1))[1];
    cout << "\n(*(pNumbers+1))[2] = " << (*(pNumbers+1))[2];
    return 0;
}
```

Result ????

Allocating and Disposing of memory

- Use New operator to allocate memory for a pointer variable
  int *pNum = new int;
  double *Distance = new double[12];

- Use delete operator to free memory allocated for a pointer variable

```
#include <iostream>
using namespace std;
int main()
{
	int number[] = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
	int *pNumbers = number;
	int numberOfMembers = sizeof(number) / sizeof(int);
	cout << "List of Numbers";
	for(int i = 0; i < NumberOfMembers; i++)
		cout << "\nNumber " << i + 1 << ": " << *(pNumbers+i);

	delete [] pNumbers;
	pNumbers = NULL;
	return 0;
}
```

Single Dimensional Arrays and Functions

```
#include <iostream>
using namespace std;
int SumOfNumbers(int Nbr[], int Size) {
   int Sum = 0;
   for(int i = 0; i < Size; i++)
        Sum += Nbr[i];
   return Sum;
}
int main() {
   int number[] = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30,
   31 };
   int numberOfMembers = sizeof(Number) / sizeof(int);
   int Value = SumOfNumbers(number, numberOfMembers);
   cout << "Sum of numbers: " << Value;
   return 0;
}
```

Result ????

Multi-Dimensional Arrays and Functions

```
#include <iostream>
using namespace std;
void DisplayNumbers(int *Nbr[], int r, int c);
int main() {
    int number[2][6] = { { 31, 28, 31, 30, 31, 30 }, { 31, 31, 30, 31, 30, 31 } };
    int *pNumbers[2];
    *pNumbers = number[0];
    for(int i = 0; i < 6; i++) {
          (*pNumbers)[i] = number[0][i];
    }
    *(pNumbers+1) = number[1];
    for(int i = 0; i < 6; i++) {
          (*(pNumbers+1))[i] = number[1][i];
    }
    cout << "List of Numbers";
    DisplayNumbers(pNumbers, 2, 6);
    return 0;
}
void DisplayNumbers(int *nbr[], int rows, int columns) {
    for(int i = 0; i < rows; i++) {
        for(int j = 0; j < columns; j++) {
            cout << "\nNumber[" << i << "][" << j << "]: " << (*(nbr+i))[j];
        }
    }
}
```

Result ????

Pointer and Function

- Declaring a pointer to function:
  Syntax:
  DataType **(**FunctionName*)();**

```
#include <iostream>
using namespace std;
int Addition(int a, int b) {
      return a + b;
}
int main() {
      int (*SomeNumber)(int x, int y);
      int x = 128, y = 5055;
      // Assign the MovieQuote() function to the pointer to function
      SomeNumber = Addition;
      // Call the pointer to function as if it had been defined already
      cout << x << " + " << y << " = " << SomeNumber(x, y);
      return 0;
}
```

Data Input and Output

- Data input:
  Function: cin
- Data output/display:
  Function: cout

CIN

- Data input in C++ is performed using the **cin** operator (cin also is a class). The cin is configured with so much flexibility (in C++, we consider that it is overloaded; in reality, it is the >> operator that is overloaded) that it can retrieve any type of declared regular variable.

```
#include <iostream>
#include <conio>
using namespace std;
#pragma hdrstop
#pragma argsused
int main(int argc, char* argv[]) {
      cout << "Enter a natural number: ";
      int Natural;
      // Retrieve an integer
      cin >> Natural;
      double Precision;
      // Retrieving a double-precision number
      cin >> Precision;
      cout << "Are you ready for C++ (y=Yes/n=No)? ";
      char Answer;
      // Requesting a character
      cin >> Answer;
      return 0;
}
```

COUT

- The values and expressions used in C++ are displayed using the **cout** extractor. To make the displaying of data more realistic, the cout is configured to handle or format data to any desired result.

```
#include <iostream>
#include <conio>
using namespace std;
#pragma hdrstop
//---------------------------------------------------------------------------
#pragma argsused
//---------------------------------------------------------------------------
int main(int argc, char* argv[]) {
      double Value1 = 6.5, Value2 = 25.56, Value3 = 28478.12;
      cout.width(16);
      cout << Value1 << endl;
      cout << setw(8) << Value2 << endl;
      cout << setw(12) << Value3 << endl;
      cout << "\nPress any key to continue...";
      getch();
      return 0;
}
```

Exception handling

```
try {
    // Try the program flow
} catch(Argument) {
   // Catch the exception
}
```

- The **try** keyword is required. It lets the compiler know that you are anticipating an abnormal behavior and will try to deal with it.
- **catch(***Argument***) {***WhatToDo***}** This section always follows the **try** section and there must not be any code between the **try**'s closing bracket and the **catch** section. The **catch** keyword is required and follows the **try** section. The **catch** behaves a little like a function. It uses an argument that is passed by the previous try section. The argument can be a regular variable or a class. If there is no argument to pass, the **catch** must at least take a three-period argument as in **catch(…)**
-

Exception handling - Example

```
#include <iostream.h>
int main() {
   int StudentAge;
   try {
        cout << "Student Age: ";
        cin >> StudentAge;
        if(StudentAge < 0)
                throw "Positive Number Required";
        cout << "\nStudent Age: " << StudentAge << "\n\n";
   } catch(const char* Message) {
        cout << "Error: " << Message;
   }
   cout << "\n";
   return 0;
}
```

Shallow copy

- Shallow copy
  - Using the standard assignment operator.
    -> Create another reference to the original object.
  - Cause run-time errors, especially with the creation and deletion of objects.

Shallow copy

```
struct Student {
        int mark;
};

void shallowCopy(Student src, Student dest) {
        dest.mark = src.mark;
}

void main() {
        Student src;
        Student dest;
        src.mark = 10;
        shallowCopy(src, dest); // ok
}
```


Thực hành trên class

Shallow copy

```
struct Student
{
        char *name;
};
void shallowCopy(Student src, Student dest)
{
        dest.name = src.name;
}
void main()
{
        Student src;
        Student dest;
        src.name = malloc(5);
        src.name = "12345";
        shallowCopy(src, dest);
        delete[] src.name;      // ok
        delete[] dest.name;     // error here
}
```


Thực hành trên class

Deep copy

- Deep copy
  - Copy everything from an object to create a whole new object.

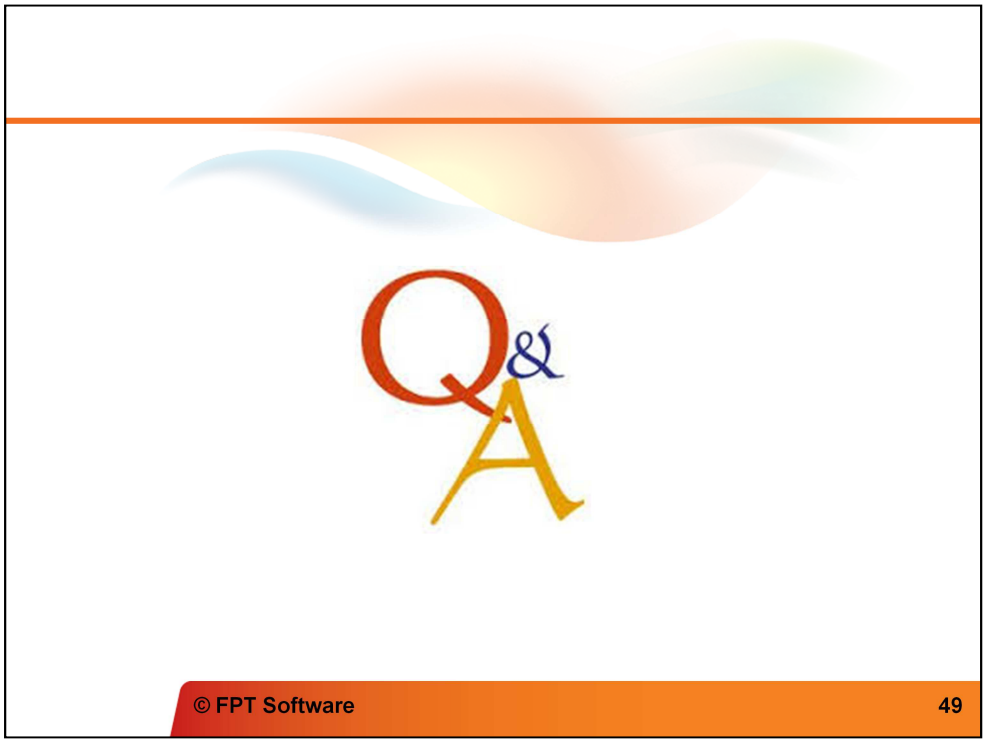
  -> The copied object can be whatever it wants.

  -> The object that was copied from will not be affected.
  - Write own copy constructors and overloaded assignment operators.

Deep copy

```
void deepCopy(Student src, Student dest) {
        if (!dest.name) {
            delete[] dest.name;
        }
        if (src.name) {
            dest.name = malloc(strlen(src.name)+1);
            memcpy(dest.name, src.name);
        }
}
void main() {
        Student src;
        Student dest;
        src.name = malloc(5);
        src.name = "12345";
        deepCopy(src, dest);
        delete[] src.name;        // ok
        delete[] dest.name;  // ok
}
```


Thực hành trên class


Q&A