fit@hcmus

Object-Oriented Programming

General Programming

Tran Duy Hoang

Topics covered

- **Coding Convention**
- Function Overloading
- Function Template
- Function Pointer

Coding Convention

- A set of guidelines for writing code
- Help improve
  - code readability
  - maintainability
  - collaboration
- Note that: No universal standard, depends on
  - programming languages, companies, communities

Naming Conventions

- Variables
  - Use ***snake_case*** or ***camelCase*** for variable names
  - Use ***UPPER_CASE*** for constants
  - Use descriptive names

Should	Should not

int price, tax, total;
int total_count = 10;
int totalCount = 10;
const double PI = 3.14159;
	int x, y, z;
	int TotalCount = 10;
	int TOTAL_COUNT = 10;
	const double pi = 3.14159;

Naming Conventions

- Functions
  - Use ***snake_case*** or ***camelCase***
  - Function names should be verbs

Should	Should not

int calculateDistance();
void calculateArea();
int getTotalUsers();
	int distance();
	void Calculate_area();
	int Gettotalusers();

Naming Conventions

- Classes & Structs
  - Use ***PascalCase*** for class and struct names
  - Use nouns for class names

Should	Should not

class UserAccount {
    // Class definition
};
	class user_account {
	    // Class definition
	};

Statement Convention

- Use spaces around operators

Should	Should not
x = a + b – c * d; for (int i = 0; i < n; i++);	x=a+b-c*d; for(int i=0;i<n;i++);

- Write one statement on one line

Should	Should not

int a;
float b;
if (a > 10)
  b = 5;
	int a; float b;
	if (a > 10) b = 5;

Statement Convention

- Group related statements in paragraph

Should	Should not

a = 5;
b = 6;
 	
if (a > b)
  max = a;
	a = 5;
	b = 6;
	if (a > b)
	  max = a;

Statement Convention

- Indent statement blocks

Should	Should not

if (a[j] > a[i])
{
  int temp = a[i];
  a[i] = a[j];
  a[j] = temp;
}
	if (a[j] > a[i])
	{
	int temp = a[i];
	a[i] = a[j];
	a[j] = temp;
	}

- Split long function (> 10 statements) into smaller ones

Comment Convention

- Explain each function with comments

Should	Should not
// This function sum up two input integers int sum(int a, int b) { }	int sum(int a, int b) { }

- Comment to complex if/loops/expressions when possible

Should	Should not

// Find max between a and b
max = (a > b) ? a : b;
 	
// Calculate x^n
for (int i = 0; i < n; i++)
  s = s * x;
	max = (a > b) ? a : b;
	
	for (int i = 0; i < n; i++)
	  s = s * x;

Comment Convention

- Doxygen-style comments for functions and classes

```
/**
 * @brief Calculates the square of a number.
 * @param x The number to be squared.
 * @return The square of x.
 */
int square(int x)
{
    return x * x;
}
```

Topics covered

- Coding Convention
- **Function Overloading**
- Function Template
- Function Pointer

Function Overloading

- Allow multiple functions to have
  - the same name
  - different parameters
- Improve code readability and reusability
- Example:
  - double sort(int a[ ], int size);
  - double sort(float a[ ], int size);

Function Overloading

- Function Signature/Prototype
  - Help the compiler determine which function to call
  - Consist of
    - Function name
    - Parameter list
    - Return type

Return type is not part of the signature for overloading decisions

Function Overloading

- Invalid function overloading?
  1. int add(int a, int b);
  2. int add(int x, int y);
  3. int add(int a, float b);
  4. float add(int a, int b);
  5. int add(float a, int b);
  6. int add(int a, int b, int c);

Topics covered

- Coding Convention
- Function Overloading
- **Function Template**
- Function Pointer

Function Template

- Allow a function to work with any data type

```
// Function template
template <typename T>
T add(T a, T b)
{
    return a + b;
}
```

```
int main()
{
    cout << add(5, 10) << endl;     // Works for int
    cout << add(3.5, 2.5) << endl;  // Works for double
    return 0;
}
```

Function Template

- Note that:
  - The keyword "***template***" can be replaced by "***class***"
  - Template declaration must be included in
    - function declaration
    - function implementation
  - Function implementation must be in the same file with
    - function declaration
    - main() function

Topics covered

- Coding Convention
- Function Overloading
- Function Template
- **Function Pointer**

Function Pointer

- Allow calling functions dynamically
- Pass functions as arguments

```
int calc(int x, int y, int (*p) (int, int))
{
    x = x * x;
    y = y * y;
    return p(x, y);
}
```

```
int add(int x, int y) {
    return x + y;
}
int mul(int x, int y) {
    return x * y;
}
int main() {
    int x = calc(2, 3, add);
    int y = calc(2, 3, mul);
}
```

Function Pointer

- Use "***typedef***" or "***using***" improves readability
  - Pass function pointer directly

    int calc(int x, int y, **int (*p) (int, int)**) {}

  - Use ***typedef***

    typedef int (***Operation**) (int, int);

    int calc(int x, int y, **Operation** p) {}

  - Use ***using***

    using **Operation** = int (*) (int, int);

    int calc(int x, int y, **Operation** p) {}

Function Pointer

- Function pointer with function template
  - Pass function pointer directly

```
template <typename T>
T calc(T x, T y, T (*p) (T, T)) {}
```

Exercise

- Exercise 1.1
  - Implement a sort function that can sort an array of any type in ascending order. For example, the function should be able to sort int[], float[], char[], and more.
  - Hint: use function template

Exercise

- Exercise 1.2
  - Implement a sort function that can sort an array of integers, allowing the user to dynamically choose between ascending and descending order.
  - Hint: use function pointer

Exercise

- Exercise 1.3
  - Implement a sort function that can sort an array of any type, allowing the user to dynamically choose between ascending and descending order.
  - Hint: use function pointer with function template